Diplomarbeit

Vergleich von Round-Robin und relationalen Datenbanken als Speicher von Nagios Performance Daten

ausgeführt zum Zweck der Erlangung des akademischen Grades eines

„Diplom-Ingenieurs für technisch-wissenschaftliche Berufe“

am Masterstudiengang Telekommunikation und Medien
der Fachhochschule St. Pölten

unter der Erstbetreuung von

Dipl.-Ing. Hans Mühlehner

Zweitbegutachtung von

Ing. Mag. Helmut Kaufmann MAS

ausgeführt von

Jan Hehenberger
tm0710262019

Ort, Datum Unterschrift

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Diplomarbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe.

- ich dieses Diplomarbeitsthema bisher weder im Inland noch im Ausland einem Begutachter oder einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Diese Arbeit stimmt mit der von den Begutachter/inne/n beurteilten Arbeit überein.

Ort, Datum Unterschrift

Zusammenfassung

Ziel dieser Arbeit ist die Ausarbeitung der Vor- und Nachteile von Round-Robin und relationaler Datenbank als Speicher von Nagios Performance-Daten. Die Arbeit gibt eine kompakte Einführung in die Welt des Network-Monitoring mit Nagios und stellt danach verschiedene AddOns vor, welche jeweils eines der beiden Datenbankkonzepte als Grundlage verwenden. Dies ist eben der Hintergrund bzw. der Beweggrund für diese Arbeit. Die AddOns verwenden entweder eine relationale Datenbank (RDB) oder eine Round-Robin Datenbank (RRD) als Performance-Daten Speicher. Beide Konzepte haben ihre Kritikpunkte und ihre Stärken.

Um diese Vor- und Nachteile in funktionaler Hinsicht zu vergleichen, werden die Anforderungen, welche an einen Performance-Daten Speicher gestellt werden, zusammengefasst aufgestellt und beide Konzepte hinsichtlich dieser Anforderungen verglichen. Den zweiten Schwerpunkt dieser Arbeit bildet der Vergleich der Performance, also welche Zeit wird benötigt um die Daten, welche in großer Anzahl und hoher Frequenz auftreten können, abzulegen.

Die Ergebnisse zeigen, dass keine eindeutige Empfehlung auszusprechen ist. Auch deshalb, weil es derzeit keine aktuelle und fertige Lösung, welche auf einer relationalen DB basiert, gibt. Eine Softwarelösung, basierend auf einer RDB bietet jedoch in vielerlei Hinsicht mehr Flexibilität und Möglichkeiten. Um zu solch einer Lösung zu gelangen, ist jedoch, im Gegensatz zu einer Entwicklung basierend auf einer RRD, ein beträchtlicher Aufwand notwendig. Die Firma NETWAYS entwickelt derzeit den NETWAYS Grapher v2 basierend auf einer RDB. Diese Produktentwick-

lung birgt viel Potential, jedoch ist abzuwarten ob die Performance auch in einem datenintensiven Umfeld ausreichend ist. Denn die Experimente in dieser Arbeit haben gezeigt, dass die RRD der RDB hinsichtlich Performance eindeutig überlegen ist.

Abstract

The present work describes the comparison of Round-Robin and relational databases as storage for Nagios-Performance Data.

This investigation introduces Nagios as a tool for Network-Monitoring and presents various AddOns, that use one of the two database concepts as storage space: either the concept of the Relational Database (RDB) or the Round-Robin Database (RRD). It is the goal of the present work to compare both concepts and discuss their specific advantages and disadvantages.

To perform this comparison, first the requirements expected from this kind of database were specified and, based on these specifications, the databases were compared.

In a second part of this work both concepts were compared based on their performance. The performance of a database describes the time a database needs for saving the data that can occur with high frequency and number.

According to the results of this investigation it is not possible to explicitely prefer one database over the other. Software solutions basing on RDB offer in general more flexibility and a wider range of possibilities than those based on RRD, but hold the drawback that there is no fully developed concept for relational databases available at the moment. In contrast to RRD based software solutions, a complete and up to date solution based on RDB requires a considerable amount of developmental effort. The company NETWAYS is currently working on the development of the NETWAYS Grapher 2, a database using the concept of the RDB. This project seems to be very

promising, but only the application in a data intensive environment will show if it will actually meet the high expectations. This is of special interest, as the RRD are superior to the RDB regarding to their performance.

Inhaltsverzeichnis

Kapitel 1

Einführung

Diese Arbeit beschäftigt sich mit dem Thema des Network Monitoring bzw. mit Teilgebieten davon. Eine der bedeutensten Softwarelösungen für diese Zwecke ist *Nagios*. Auf Grund der allgemeinen Bedeutsamkeit dieser Software wird die Herangehensweise, an die in dieser Arbeit behandelten Themen, mit überwiegendem Augenmerk auf Nagios erfolgen.

Der Schwerpunkt der Arbeit liegt vor allem auf einem bestimmten Bereich des Network Monitoring, dem Generieren von Reports und im speziellen dem *Graphing und Trending* von Performance Daten. Im ersten Abschnitt der Arbeit wird zunächst jedoch auch ein allgemeiner theoretischer Hintergrund zu Network Monitoring mit Nagios und dem Schwerpunktthema Graphing und Trending von Performance Daten vermittelt.

Da Nagios selbtst keine Möglichtkeit bietet Performance Daten abzuspeichern und darzustellen, gibt es verschiedene Arten von Addons welche für Graphing und Trending verwendet werden können. Diese Ansätze verwenden teilweise eine RRD (Round Robin Database) oder eine herkömmliche relationale Datenbank. Es soll gezeigt werden, wo die Vor- und Nachteile der beiden Herangehensweisen liegen. Sowohl in Hinblick auf Features als auch unter Beachtung der Performance.

Die Forschungsfrage, welche sich daraus ergibt, lautet folgendermaßen:

- Was sind Vor- und Nachteile der Konzepte RRD und relationale Datenbank, unter besonderem Augenmerk auf die Speicherung von Performance Daten?

Um die Anforderungen, welche die Arbeit stellt, erfüllen zu können, sind verschiedene Forschungsstrategien geplant. Grundlage der ersten Abschnitte bildet eine intensive Literaturrecherche, um das notwendige Hintergrundwissen zu den bereits erwähnten Themen zu erarbeiten.

Auch für den Hauptteil, in welchem die Konzepte RRD und relationale DB näher betrachtet werden, wird zunächst eine Literaturrecherche von Nöten sein. Hauptaugenmerk liegt in diesem Teil der Arbeit jedoch auf dem praktischen Test der beiden Ansätze, welcher in Form eines Experiments durchgeführt wird, um die Vor- und Nachteile detailliert herauszuarbeiten.

Kapitel 2

Begriffserklärungen

2.1 Network Monitoring

"Unter Netzwerk-Monitoring versteht man die Überwachung und regelmäßige Kontrolle von Netzwerken, deren Hardware (z. B. Server, Router, Switches) und Diensten (z. B. Webserver, DNS-Dienste, E-Mail-Dienste)."[1]

Tatsächlich wird beim Network Monitoring (auf Deutsch: Netzüberwachung) also nicht nur das Netzwerk an sich, sondern auch die Geräte und Dienste, welche in diesem vorkommen, überwacht. Der genaue Umfang dieser Möglichkeiten hängt aber natürlich von der eingesetzten Monitoring Lösung ab.

2.2 Performance Daten

Die Überwachung von Hosts (alle im Netzwerk vorkommenden Geräte) und Services wird nicht durch interne Mechanismen von Nagios selbstständig durchgeführt, sondern durch dessen Plugins. Diese Plugins liefern dann zumindenst einen Return-Code und eine Statusinformation an Nagios zurück. Die Statusinformation muss mindestens aus einer Zeile, vom Menschen lesbaren Text bestehen und *kann* zusätz-

[1] wikipedia, 2008-10-26, 1, Netzwerkmanagement, http://de.wikipedia.org/wiki/Netzwerkmanagement

lich noch die Performance Daten (konkret: Plugin-Performance-Daten) beinhalten. Das vollständige Format des Performance-Daten Teils, der Statusinformationen, ist wie folgt festgelegt (Teil rechts vom Pipe-Zeichen):

```
1 <text> | <variable>=<wert><einheit>;<warn>;>crit>;<min>;<max>
```

Es wird also nicht nur der Wert einer Überwachung zurückgemeldet, sondern auch erweiterte Informationen (Schwellenwerte und Minimum bzw. Maximum). Es können pro Zeile Statusinformation auch mehrere Variablen existieren. Hauptvorteil dieser Performance Daten ist, dass diese leicht automatisiert auszuwerten sind und vielfältig benutzt werden können. Es ist vor allem erwähnenswert, dass sich Graphing und Trending Addons diese Daten zu Nutzen machen.[2]

2.3 Graphing und Trending

Graphing und Trending sind zwar verschiedene Dinge, hängen aber doch zusammen, da beide Funktionen davon abhängen, dass Performance Daten über längere Zeiträume gespeichert werden. Graphing meint einfach die grafische Darstellung der Daten in Diagrammen und Trending beschreibt die Möglichkeit, aus den über bestimmte Zeiträume gehaltenen Daten, Entwicklungen abzulesen (zum Beispiel: wie entwickelt sich der freie Plattenspeicher), um so früher auf Gegebenheiten reagieren zu können.

2.4 Round Robin Database (RRD)

"Der Vorteil von RRDs im Vergleich zu relationalen Datenbanken besteht darin, dass jüngere Daten mit einer höheren zeitlichen Auflösung als ältere gespeichert werden. Alte Daten werden durch aktuelle ersetzt. Das Resultat daraus ist eine schnelle

[2] vgl. Galstad 2007, S. 323 f

Datenbank mit fester Größe. Den Zeitraum und die Auflösung der Messdaten kann der Benutzer in so genannten RRAs (Round Robin Archives) selbst festlegen."[3]

Der Vorteil der RRD liegt also vor allem in der fixen Größe und der Schnelligkeit. Dass ältere Daten mit geringerer Auflösung gespeichert und schlussendlich ersetzt werden, ist der Preis den man dafür bezahlen muss. Wohl einer der Hauptgründe, warum *RRDtool*, eine RRD-Implementierung, häufig zur Speicherung von Performance Daten eingesetzt wird, ist das integrierte Visualisierungsfeature. Simples Graphing ist also einfach zu realisieren. Diese und weitere Aspekte werden in der Arbeit noch ausführlicher behandelt.

[3] wikipedia, 2008-10-26, 2, RRDtool, http://de.wikipedia.org/wiki/RRDtool

Kapitel 3

Hintergrund

3.1 Network Monitoring mit Nagios

3.1.1 Was ist Nagios

"Nagios ist eine System- und Netzwerküberwachungsapplikation. Sie überwacht Hosts und Services, die Sie angeben, und alarmiert Sie, wenn sich die Dinge verschlechtern und wenn sie wieder besser werden."[1]

Nagios überwacht also, wie es für eine Network Monitoring Software charakteristisch ist, Geräte im Netzwerk (bei Nagios als Hosts bezeichnet) und Services die auf diesen Hosts laufen. Zudem gibt es eine vielfältige Alamierungsfunktion. Nagios wurde für Linux entwickelt, sollte daher auch unter anderen UNIX Derivaten lauffähig sein.[2]

Die Konfiguration von Nagios ist, auch laut Handbuch, sehr komplex und erfordert viel Geduld. Da es nicht Gegenstand dieser Arbeit ist, Nagios komplett zu durchschauen und in allen Details konfigurieren zu können, wird hier vor allem auf jene Punkte eingegangen, welche für die weitere Bearbeitung der behandelten

[1] Galstad 2007, S. 4

[2] vgl. Galstad 2007, S. 4

Themen von Bedeutung sind.[3]

3.1.2 Objekte

Alle Elemente die an der Überwachungs- und Benachrichtigungslogik beteiligt sind, sind Objekte. Diese Objekte werden in Konfigurationsdateien anhand eines flexiblen Formats definiert (Pfade zu diesen Konfigurationsdateien sind in der Nagios Hauptkonfigurationsdatei anzugeben) und erleichtern einem auf lange Sicht die Nagios Konfiguration zu verwalten. So sind zum Beispiel auch Objektvererbungen möglich und sinnvoll. Die bedeutendsten Objekttypen sind:

- Hosts
- Services
- Kontakte
- Zeitfenster
- Befehle

Im folgenden werden diese Objekttypen kurz erklärt.[4]

Hosts

Hosts sind alle physikalischen Geräte im Netzwerk. Dies können zum Beispiel sein: Server, Workstations, Router, Switches, Drucker usw.

Hosts haben eine Adresse (z.B. MAC oder IP Adresse) und einen oder mehrere Services mit sich verknüpft. Weiters können Hosts in Eltern/Kind Beziehungen zueinander stehen. Diese Beziehungen werden meist verwendet um reale Netzwerkverbindungen abzubilden, um so die Erreichbarkeitslogik zu ermöglichen. Durch die *Netzwerk-Erreichbarkeits-Logik* wird eine Unterscheidung zwischen UNREACHABLE

[3] vgl. Galstad 2007, S. 13
[4] vgl. Galstad 2007, S. 80

und DOWN realisiert. Das Überwachungssystem weiß so, dass wenn ein Host (z.B. ein Switch oder Router) ausfällt (DOWN), die Hosts *hinter* diesem Switch oder Router nicht mehr erreichbar sind (UNREACHABLE).[5][6]

Die Konfiguration eines Host Objekts könnte zum Beispiel folgendermaßen aussehen:

```
1  define host{
2      host_name beispiel-host
3      alias Beispiel Host
4      address 192.168.1.254
5      parents <host-names>
6      max_check_attempts 5
7      check_period 24x7
8      contacts router-admin
9      notification_interval 30
10     notification_period 24x7
11 }
```

Es gibt noch etliche weitere Attribute, welche in der Nagios Dokumentation nachgeschlagen werden können, in diesem Beispiel sind die zumindest notwendigen Attribute angeführt. Die meisten sollten selbsterklärend sein. Das *parents* Attribute wird eben genutzt um die Eltern/Kind Beziehungen abzubilden, es werden meist die Router oder Switches angegeben, welche am nächsten zum betreffenden Host sind. Ist der Host im selben Netzwerksegment wie der überwachende Host, so wird der Wert dieses Attributs leer gelassen.

Das *max_check_attempts* Attribut ist von großer Bedeutung, da hier festgelegt wird, wie oft Nagios den Host-Prüfbefehl wiederholen soll, wenn das Plugin, welches für die Prüfung zuständig ist, einen anderen Status als *OK* liefert. Mehr zu diesen Zuständen im Abschnitt *Plugins*.

[5] vgl. Galstad 2007, S. 81
[6] vgl. Galstad 2007, S. 187 f

Bei *check_period* und *notification_period* werden Zeitfenster Objekte angegeben, bei *contacts* müssen Kontakt Objekte angegeben werden.

Ein detaillierteres Verständnis dieser und andere Attribute ist für den weiteren Verlauf dieser Arbeit vorläufig nicht notwendig.[7]

Services

Services sind die zentralen Objekte der Überwachungslogik. Sie können die Eigenschaften eines Hosts (CPU-Auslastung, freier Speicher, Plattenbelegung etc.), Services die durch den Host zur Verfügung gestellt werden (SSH, HTTP, FTP etc.) oder weitere Dinge die mit dem Host verbunden sind, repräsentieren.[8]

Die Konfiguration eines Service Objekts könnte folgendermaßen aussehen:

```
1  define service{
2      host_name beispiel-host
3      service_description check-disk-sda1
4      check_command check-disk!/dev/sda1
5      max_check_attempts 5
6      check_interval 5
7      retry_interval 3
8      check_period 24x7
9      notification_interval 30
10     notification_period 24x7
11     contacts linux-admin
12 }
```

Vor allem erwähnenswert ist hier das Attribut *check_command*. Hiermit wird das Befehls-Objekt angegeben, mit dem der Zustand des Service geprüft wird. Wiederum sind hier nur die notwendigen Attribute aufgelistet. Weitere Attribute sind in der

[7] vgl. Galstad 2007, S. 86 f
[8] vgl. Galstad 2007, S. 81

Nagios Dokumentation nachschlagbar.[9]

Kontakte

Über Kontakt Objekte werden Personen definiert, die am Benachrichtigungsprozess beteiligt sind. Kontakte haben Benachrichtigungsmethoden (SMS, Email, usw.) und erhalten Benachrichtigungen zu Hosts und Services für die sie verantwortlich sind. Siehe *contacts* bei der Host und Service Beispielkonfiguration.

Zeitfenster

In Zeitfensterobjekten kann definiert werden, wann Hosts und Services überwacht werden sollen (siehe *check_period* bei der Host und Service Beispielkonfiguration) und wann Kontakte Benachrichtigungen erhalten sollen (siehe *notification_period* bei der Host Beispielkonfiguration).

Befehle

Befehlsobjekte werden benutzt, um zu definieren, welche Programme und Skripte Nagios ausführen soll, um *Host und Service Prüfungen* oder *Benachrichtigungen* durchzuführen. Siehe *check_command* bei der Service Objekt Beispielkonfiguration.[10]

3.1.3 Plugins

Wie bereits erwähnt, wird die Prüfung von Hosts und Services nicht von Nagios selbst durchgeführt, sondern durch Plugins. Dieses Konzept macht Nagios sehr flexibel und leicht erweiterbar. Einerseits existieren bereits Plugins für die meisten Routineüberprüfungen, andererseits jedoch, hat man einen ausgefalleneren und individuellen Anspruch an eine Host oder Service Prüfung (Host oder Service Check), so kann man sich selbst das entsprechende Plugin basteln.

[9] vgl. Galstad 2007, S. 96 f
[10] vlg. Galstad 2007, S. 82

Plugins sind von der Kommandozeile ausführbare Programme oder Scripts. Die Ergebnisse dieser Plugins werden von Nagios verwendet, um den aktuellen Status eines Hosts oder Services zu bestimmen. Plugins werden beim *check_command* Attribut einer Host oder Service Definition angegeben.[11]

Return-Code

Das Plugin wird also von Nagios aufgerufen und erledigt dann seine Aufgaben, um den Zustand von Service oder Host zu ermitteln. Was jedoch alle Nagios Plugins gemein haben *müssen*, ist, dass sie mindestens eine Zeile Text an stdout liefern und mit einem bestimmten Return-Code enden. Folgende *Return-Codes* sind möglich:

0 OK

1 WARNING

2 CRITICAL

3 UNKNOWN

Dieser Return-Code ist zugleich der Service Zustand. Führt das Plugin eine Host Prüfung durch, so kann der Host Zustand nur UP, DOWN oder UNREACHABLE sein. Wobei ein Return-Code 0 UP bedeutet und die Return-Codes 2 und 3 DOWN/UNREACHABLE. Der Return-Code 1 bedeutet UP, wenn die *use_aggressive_host_checking* Option (eine Einstellung in der Nagios Hauptkonfigurationsdatei) deaktviert ist, sonst bedeutet der Return-Code 1 DOWN/UNREACHABLE. Die *use_aggressive_host_checking* Option ist standardmäßig deaktviert. Wie zwischen DOWN und UNREACHABLE unterschieden werden kann, wurde bereits ausgeführt. Dies geschieht mithilfe der *Netzwerk-Erreichbarkeits-Logik*.

[11] vgl. Galstad 2007, S. 140 f

Plugin Ausgabe

Die zweite Bedingung die ein Plugin erfüllen muss, ist, dass es eine Zeile Text ausgibt. Dies kann beliebiger, vom Menschen lesbarer, Text sein. Soll das Plugin jedoch auch Performance Daten liefern, so muss es sich an folgende, bereits erwähnte, Konvention bei der Ausgabe halten.

```
1 TEXT-OUTPUT | PERFORMANCE DATA
```

Wobei sowohl Text als auch Performance Daten mehrzeilig sein können.

Vorraussetzung an ein Nagios Plugin ist also nur, dass es sich an die Konventionen hinsichtlich Return-Code und Ausgabe hält, die internen Abläufe des Plugins, sind für Nagios nicht von Bedeutung.[12]

3.1.4 Verarbeitung von Performance Daten

Grundsätzlich gibt es bei Nagios zwei Arten von Performance Daten. *Prüf-Performance-Daten* und *Plugin-Performance-Daten*. Bei der bereits besprochenen Art von Performance Daten, also jene, welche in der Ausgabe eines Plugin beinhaltet sein können, handelt es sich um Plugin-Performance-Daten.

Prüf-Performance-Daten

Dies sind interne Daten, die direkt von Nagios zur Verfügung gestellt werden und sich auf die Ausführung einer Host- oder Serviceprüfung beziehen. Typische Daten sind z.B. wie lange die Ausführung der Serviceprüfung dauerte oder die Service-Prüfverzögerung (wieviel Zeit verging von der geplanten Ausführung bis zur tatsächlichen Ausführung).

Entnommen werden können diese Daten verschiedensten Makros. Für die angeführten Beispiele wären das die *$SERVICEEXECUTIONTIME$* und die *$SERVICELATENCY$* Makros.

[12] vgl. Galstad 2007, S. 323 f

Plugin-Performance-Daten

Der Performance-Daten-Teil der Plugin Ausgabe, sollte folgendes Format aufweisen:

```
1 <variable>=<wert><einheit>;<warn>;>crit>;<min>;<max>
```

Wobei eine Zeile auch mehrere Variablen beinhalten kann (durch Leerzeichen getrennt).

Klarzustellen ist, das Format der Performance Daten ist nicht eindeutig definiert, da Nagios diese Daten selbst nicht weiterverarbeitet. Das angegebene Format kommt aber häufig zur Anwendung und ist in den *Nagios plug-in development guidelines* so festgehalten. Erwartet wird es zum Beispiel vom *NagiosGrapher* so. Welches Format das verwendete Addon aber tatsächlich verarbeitet, ist dessen Beschreibung zu entnehmen.

Der gesamte Performance-Daten-Teil wird dann, je nachdem ob es sich um eine Host- oder Serviceprüfung handelt, in den *$HOSTPERFDATA$* oder *$SERVICEPERFDATA$* Makros gespeichert. Jedoch muss, damit Performance Daten verfügbar sind und verarbeitet werden können (sowohl Plugin- also auch Prüf-Performance-Daten), die *process_performance_data* Option in der Hauptkonfigurationsdatei gesetzt werden.

Performance Daten können dann weiterverarbeitet werden, indem sie entweder in Dateien gespeichert oder indem Befehle ausgeführt werden, welche die Daten weiterverarbeiten oder an externe Applikationen (z.B. Graphing und Trending Addons) weiterleiten. Diese Befehle (Befehls Objekte) oder Dateien (Dateipfade) werden ebenfalls durch Optionen in der Hauptkonfigurationsdatei angegeben. Es handelt sich dabei um die Optionen *host_perfdata_command*, *service_perfdata_command*, *host_perfdata_file* und *service_perfdata_file*.

Der Vorteil der Variante, die Daten in Dateien zu schreiben, ist ein geringerer CPU-Overhead. Also vor allem, wenn die Daten von vielen Host- und Serviceprüfungen verarbeitet werden müssen, so ist diese Variante durchaus in Betracht zu ziehen.

Die Performance Daten werden also von Graphing und Trending Addons verwendet. Liefert ein Plugin keine Performance Daten, so können die Addons natürlich auch auf die reine Textausgabe zurückgreifen (was manche Addons auch tatsächlich machen), aber besser verwertbar sind natürlich die Performance Daten.

Wird im weiteren Verlauf der Arbeit von *Performance Daten* gesprochen, so sind, ausser es ist ausdrücklich anders erwähnt, die *Plugin-Performance-Daten* gemeint.[13][14]

3.1.5 Statustypen

In den vorangegangenen Abschnitten wurden bereits die verschiedenen Zustände besprochen, in denen ein Service oder Host sein kann. Der Status eines Services oder Hosts wird aber noch durch einen weiteren Parameter bestimmt, dem Statustyp. Bei Nagios gibt es genau zwei Statustypen: SOFT und HARD. Wann ein Status HARD oder SOFT ist, hängt von einigen Faktoren ab. Entscheidende Rolle spielt das bereits besprochen Attribut von Services und Hosts *max_check_attempts*. Wird z.B. ein Host oder Service überprüft und das Ergebnis ist nicht-UP oder nicht-OK und die Prüfung wurde noch nicht so oft wiederholt, wie es mit *max_check_attempts* angegeben wurde, so ist der Statustyp SOFT. Für die Bestimmung des Statustyps gibt es aber noch eine Reihe weiterer Richtlinien, deren Erläuterung hier jedoch nicht notwendig ist.

Viel wichtiger ist, was es bedeutet, ob ein Status HARD oder SOFT ist. Handelt es sich um einen SOFT-Status, so wird der SOFT-Status protokolliert und die Eventhandler zur Behandlung des SOFT-Status aufgerufen. Diese Eventhandler können genutzt werden, um eine proaktiver Fehlerbehandlung durchzuführen. Also das Problem zu beheben, bevor der Status als HARD eingestuft wird. Ist ein Service oder Host in einem HARD-Status so wird dies ebenfalls protokolliert, der Evenhandler zur Behandlung des HARD-Status aufgerufen *und* die Kontakte welche in

[13] vgl. Galstad 2007, S. 259 f

[14] vgl. nagios-wiki, 2008-10-28, 1, NagiosGrapher,
http://www.nagios-wiki.de/nagios/howtos/nagiosgrapher

der Host bzw. Service Konfiguration eingetragen sind informiert.[15]

3.2 Graphing und Trending

Es ist naheliegend, dass man die von den Plugins ermittelten Daten grafisch darstellen möchte. Die Information ist in den Diagrammen leichter zu erfassen und es lassen sich Entwicklungen über die Zeit (Trends) leicht erkennen.

3.2.1 Graphing und Trending Lösungen

Es existieren eine Reihe von Addons, welche dem Graphing von Plugin-Performance-Daten dienen. Eine Liste dieser Addons ist unter *nagiosexchange.org* unter *Utilities - Addon Projects - Charts* zu finden. Aktuell sind ungefähr 14 Addons vorhanden welche Performance Daten speichern und/oder diese grafisch darstellen. Die bedeutensten davon, werden in diesem Abschnitt vorgestellt.

Der, für diese Arbeit, wichtigste Unterschied zwischen den verschiedenen Addons ist, dass manche Round Robin Datenbanken und andere relationale Datenbanken zur Speicherung der Performance Daten verwenden. Die Eröterung der Vor- und Nachteile dieser Konzepte ist Gegenstand dieser Arbeit und wird in Kapitel 4 behandelt. Im Abschnitt 3.1 auf Seite 22 ist eine Übersicht, welche Addons, sich welches Konzept zu Nutzen machen.

Für die weitere Betrachtung werden nur Addons ausgewählt, welche eine Gesamtlösung anbieten (sowohl Speicherung als auch Darstellung der Daten) und aktuell noch von Bedeutung sind. Die Addons werden kurz vorgestellt um einen gewissen Überblick über die verfügbaren Lösungen zu bekommen.[16]

[15] vlg. Galstad 2007, S. 181 f

[16] vgl. nagiosexchange, 2008-11-05, 1, NagiosExchange: Utilities/Addon Projects/Charts, http://www.nagiosexchange.org/cgi-bin/page.cgi?g=Utilities%2FAddOn_Projects%2FCharts%2Findex.html;d=1

Bezeichnung	Datenbank	Letzte Version vom	Funktion
NagiosGrapher	RRD	04.06.2008	Speichern und Darstellen
NETWAYS Grapher v2	objektrel. DB	11.09.2008	Speichern und Darstellen
APAN	RRD	16.12.2003	Speichern und Darstellen
BrainyPDM	objektrel. DB	24.07.2008	Speichern und Darstellen
FIFO-RRD / RRD-Graph	RRD	25.11.2005	Speichern und Darstellen
mypan	RRD	24.03.2003	Speichern und Darstellen
n2rrd	RRD	28.07.2002	Speichern
nagiosgraph	RRD	01.10.2008	Speichern und Darstellen
nagiostat	RRD	17.07.2004	Speichern und Darstellen
perf2rrd	RRD	22.10.2007	Speichern
PerfData	RRD	keine Information	Speichern und Darstellen
PerfParse	objektrel. DB	11.04.2006	Speichern und Darstellen
PNP4Nagios	RRD	27.10.2008	Speichern und Darstellen
SmiStat	RRD	27.05.2005	Speichern und Darstellen

Tabelle 3.1: Vergleich verschiedener Graphing Addons

NagiosGrapher

Der NagiosGrapher ist eine von der NETWAYS GmbH entwickelte Open-Source Software. Der NagiosGrapher übernimmt die Performance Daten der Plugins und speichert diese in einer RRD ab. Das Addon verwendet nur Standardschnittstellen, es sind daher keine Anpassungen an Nagios selbst erforderlich. Die Diagramme, welche bei Aufruf immer auf Grund der aktuellen Daten der RRD erzeugt werden, können direkt aus dem Nagios Interface abgerufen werden.

Weiters werden neue Host und Service Objekte automatisch von NagiosGrapher erkannt und konfiguriert und die Performance Daten können optional auch zusätzlich in einer relationalen Datenbank abgelegt werden, damit die Daten nicht dem automatischen Überschreiben der RRD unterliegen.[17][18]

NETWAYS Grapher v2

Der NETWAYS Grapher v2 ist die neue Version des Graphers aus dem Hause NETWAYS. Er unterscheidet sich stark von der ersten Version, dem NagiosGrapher, und soll laut NETWAYS "vor allem im Bezug auf Darstellung, Nutzerfreundlichkeit und Datengenauigkeit und -flexibilität neue Maßstäbe setzen."[19]

Neben den Verbesserungen hinsichtlich Darstellung und Benutzerfreundlichkeit, welche durch den Einsatz von Flash und Ajax erreicht wurden, sind vor allem die Veränderunge bezüglich Datenspeicherung und Datenmanagement erwähnenswert. Die Daten werden nun nicht mehr, wie beim NagiosGrapher, in einer RRD, sondern eben in einer objektrelationalen DB gespeichert. Dieser Trend, von der RRD zur relationalen DB ist bemerkenswert und die Gründe dafür, werden im Kapitel 4 untersucht.

[17] vgl. wikipedia, 2008-11-12, 3, NagiosGrapher, http://de.wikipedia.org/wiki/NagiosGrapher

[18] vgl. netways, 2008-11-12, 1, Performance Charts mit NagiosGrapher, http://www.netways.de/de/produkte/nagios_addons/nagiosgrapher

[19] netways, 2008-11-12, 2, NETWAYS Grapher v2, http://www.netways.de/de/produkte/nagios_addons/netways_grapher_v2/

Weiters wurde die Darstellung und Anwendung der *Multigraphs* (mehrere Diagramme in einem Fenster um Korrelationen zu finden) verbessert.[20]

BrainyPDM

BrainyPDM ist noch in einem sehr frühen Entwicklungsstadium, daher ist auch nur eine Alpha-Version verfügbar. Dieses Open-Source Addon wurde und wird in Java 1.5 entwickelt. Die Diagramme werden periodisch in einem festgelegten Zeitintervall erzeugt und als PNG Dateien abgelegt. Die maximal stündliche Diagrammerzeugung und, dass die Diagramme nicht über ein eigenes oder das Nagios Interface zugänglich sind, sind schwerwiegende Einschränkungen, welche die Entwickler auf jeden Fall noch aufheben sollten, wollen sie mit ähnlichen Addons konkurrieren.

Interessanter Ansatz ist, dass die Daten in einer relationalen DB abgelegt werden und nur zur Diagrammerzeugung in eine RRD übertragen werden.

In der nächsten Version (2.0) sollen laut Entwickler die Features *BrainyPDM custom graph* und *BrainyPDM prediction* enthalten sein. *BrainyPDM prediction* lässt eine automatische Analyse von Trends vermuten, genauere Information sind dazu jedoch nicht verfügbar. Da laut Roadmap am 10. Oktober 2008 die erste Beta-Version (1.0 beta) erscheinen hätte sollen (was nicht geschehen ist) und keine weiteren Ankündigungen gemacht wurden, sind zumindest leichte Zweifel am weiteren Fortschritt dieses Projekts und ob jemals eine Version 2.0 erscheinen wird, angebracht.[21][22][23]

nagiosgraph

nagiosgraph ist eine Sammlung von Skripts zur Speicherung und Darstellung von Performance Daten. Ziel der Entwicklung von *nagiosgraph* war es, ein möglichst

[20] vgl. netways, 2008-11-12, 2, NETWAYS Grapher v2,
http://www.netways.de/de/produkte/nagios_addons/netways_grapher_v2/

[21] vgl. Bagari, 2008-11-12, 1, Automatic graph based on RRD,
http://www.brainypdm.org/index.php/documentation/4-architecture-documentation/13-rrdgraph

[22] vgl. Bagari, 2008-11-12, 2, Roadmap, http://www.brainypdm.org/index.php/roadmap

[23] vgl. Bagari, 2008-11-12, 3, Features, http://www.brainypdm.org/index.php/features

einfaches Addon zur Verfügung zu stellen. Die Einfachheit wird gewährleistet durch einen beschränkten Funktionsumfang, geringe Konfigurationsmöglichkeiten und indem, wie auch beim NagiosGrapher oder beim NETWAYS Grapher v2, neue Hosts und Services automatisch erkannt werden.

Die wichtigsten Bestandteile sind das *insert.pl* Skript, welches die Performance Daten in die RRD überträgt und das *show.cgi* Skript, welches eine HTML Seite mit den Diagrammen zu den spezifizierten Hosts und Services erstellt. In der *map* Datei wird festgelegt welche Daten in welcher RRD Datei gespeichert werden sollen. Die Skripts *insert.pl* und *show.cgi* greifen auf die *map* Datei zurück.[24]

PerfParse

PerfParse liest die Performance Daten aus den Dateien, welche in der Hauptkonfigurationsdatei als Speicher für die Performance Daten angegeben wurden. Der Parser *PerfParse*, welcher dem Addon seinen Namen gibt, bereitet die Daten auf und schreibt diese in eine MySQL Datenbank. Mittels des Tools *perfchart* (Natürlich ein Teil von PerfParse) können dann die Diagramme erstellt werden. Die Diagramme werden in Form von PNG Dateien erzeugt.

Natürlich können die Daten, welche ja in einer relationalen Datenbank (MySQL) stehen, auch von anderen Tools gelesen werden, deren Einsatzzweck vielfältig sein kann. Man ist also keineswegs an die Darstellungs-Komponenten von *PerfParse* gebunden.

PefParse wurde leider schon lange nicht mehr weiterentwickelt und wie es mit dem Projekt weitergeht, ist noch unklar. Die aktuell verfügbare Version wurde nicht für die Nagios Version 3.0 oder neuer entwickelt.[25]

[24] vgl. Dossing, 2008-11-12, 1, nagiosgraph README,
http://nagiosgraph.cvs.sourceforge.net/*checkout*/nagiosgraph/nagiosgraph/README

[25] vgl. perfparse, 2008-11-12, 1, PerfParse, http://perfparse.de/tiki-index.php?page=PerfParse

PNP4Nagios

Bei *PNP4Nagios* gibt es drei Möglichkeiten, um die Performance Daten in die RRD zu übertragen. Im *Default Mode* wird für jeden Service oder Host Check automatisch das *process_perfdata.pl* Skript aufgerufen (Also Performance Daten Verarbeitung auf *command* Basis). Dieser Mode funktioniert bei geringem Überwachungsumfang und niedriger Frequenz der Checks (ungefähr 5 Minuten) zufriedenstellend. Im sogenannten *Bulk Mode* werden die Performance Daten nach einem definierbaren Zeitintervall aus der entsprechenden Datei ausgelesen (Performance Daten Verarbeitung auf *file* Basis), verarbeitet und in die RRD Dateien geschrieben. Dies geschieht in einem Schritt und für die Zeitdauer dieses Schritts ist Nagios blockiert und kann keine Service Checks machen (die Performance Daten Logdatei ist blockiert, weil geöffnet). Dies klingt auf den ersten Blick fatal, ist aber zu relativieren, da, laut den *PNP4Nagios* Entwicklern die Verarbeitung, bei ungefähr 2000 Services und einem Interval von 10 Sekunden, ca. 0,06 Sekunden dauert. Bei größerem Überwachungsumfang ist diesem Problem aber durchaus Beachtung zu schenken und der *Bulk Mode with NPCD* eine Überlegung wert. Hier wird die Performance Daten Logdatei nach Ablauf des Zeitintervalls nicht direkt verarbeitet, sondern in ein Spoolverzeichnis verschoben. Dies geschieht sehr schnell (vorrausgesetzt das Spoolverzeichnis liegt im selben Dateisystem) und die Zeitdauer der Blockierung von Nagios ist kaum erwähnenswert. Vor allem ist die Dauer unabhängig vom Überwachungsumfang. NPCD (Nagios Performance C Daemon) überwacht das Spoolverzeichnis und ruft das Verarbeitungsskript mit dem Namen der neu eingelangten Datei auf.

PNP4Nagios verarbeitet nur Performance Daten deren Format sich strikt an die Vorgaben der *Nagios plug-in development guidelines* hält.

Das *PNP4Nagios* Web Frontend (PHP) wird gestartet, indem die *index.php* aufgerufen wird. Als Parameter wird *host* und, wenn gewünscht, *srv* (Serive) übergeben. Über den *action_url* Parameter in der Host oder Service Definition wird das *PNP4Nagios* in Nagios eingebunden. Zur komfortablen Konfiguration wird man ein

Template Host und Service Objekt anlegen und Host- und Servicenamen Makros verwenden, welche dann in den konkreten Host- und Service Definitionen durch den tatsächlichen Host- bzw. Servicenamen ersetzt werden.

PNP4Nagios wird aktuell laufend weiterentwickelt, interessant wird aber sein zu beobachten, wie und ob sich die RRD basierte Lösung gegen die Datenbank basierten Lösungen, vor allem den NETWAYS Grapher v2, behaupten kann.[26][27]

3.3 Reporting mit Nagios

Die in Nagios integrierte Reporting Funktionaliät bietet nur beschränkte Möglichkeiten. Hauptaugenmerk liegt darauf, die Verteilung der Service- oder Hostzustände eines Service oder Hosts darzustellen. Die möglichen Zustände wurden bereits im Abschnitt 3.1.3 auf Seite 17 vorgestellt.

Was dem Reporting-Feature auf den ersten Blick fehlt, ist die Darstellung von Performance-Daten. Dies wird, wie bereits besprochen, von zahlreichen Addons abgedeckt.

Das Availability Reporting und weitere Reporting Möglichkeiten, welche Nagios bietet, werden über CGI Skripts realisiert. Im folgenden werden diese kurz vorgestellt.

Trends Es kann hier die Entwicklung der Service- oder Hostzustände über die Zeit dargestellt werden. Für die Darstellung kann ein einzelner Host oder Service ausgewählt werden. Es kann hier auch der Zeitraum der Darstellung definiert und weitere Einstellungen vorgenommen werden.

Availability Mit diesem CGI wird das eigentlich Availablity Reporting realisiert. Es wird die prozentuale Verteilung über die verschiedenen Zustände angezeigt. Man kann hierbei wählen ob die Verteilung der Servicezustände für einzelne

[26] vgl. pnp4nagios, 2008-11-12, 1, pnp:start, http://www.pnp4nagios.org/pnp/start
[27] vgl. pnp4nagios, 2008-11-12, 2, pnp:modes, http://www.pnp4nagios.org/pnp/modes

Services bzw. Hosts oder für Service- bzw. Hostgruppen dargestellt wird. Eine Auswahl des Zeitraums und weitere Einstellungen sind auch hier möglich. Ein beispielhafter Availability Report für ein Service ist im Abschnitt 3.1 auf Seite 29 ersichtlich.

Alert Histogram *Alert Histograms* können für Hosts und Services erzeugt werden. Es werden hier die über einen definierbaren Zeitraum aufgetretenen Alarme dargestellt. Es kann ausgewählt werden welche Ereignisse (siehe 3.1.3, Seite 17) und welche Statustypen (siehe 3.1.5, Seite 20) dargestellt werden sollen.

Alert History Hier werden einfach in Textform aufgetretenen Alarme chronologisch aufgelistet. Neben den Host und Service Alarmen werden hier auch allgemeine Ereignisse, wie zum Beispiel das Starten oder Beenden von Nagios aufgelistet.

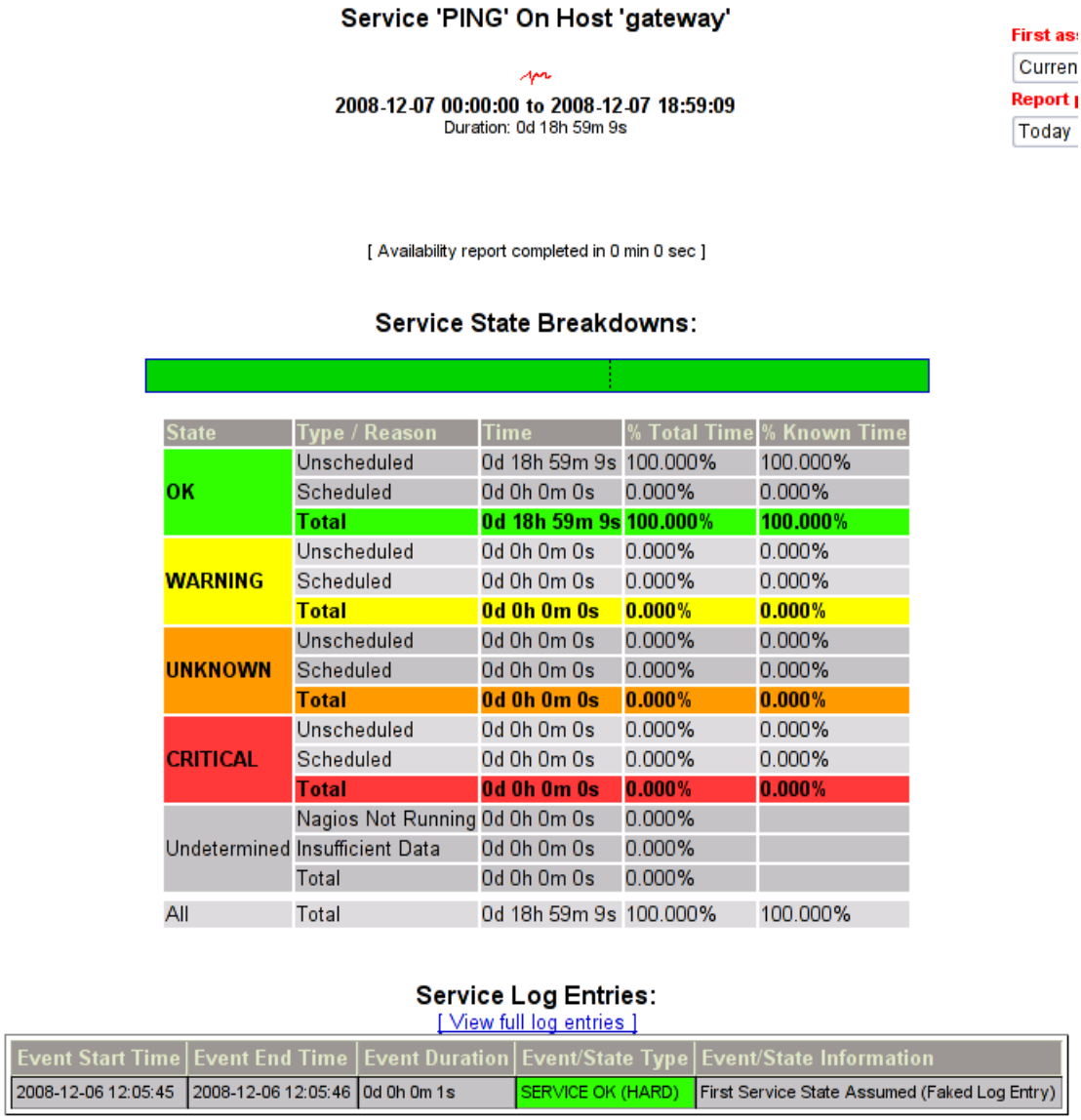
Alert Summary Hier können vordefinierte oder benutzerdefinierte *Alert Summary Reports* generiert werden. Für benutzerdefinierte Reports können, neben dem Zeitraum, auch bestimmte Hosts, Services oder Hostgruppen gewählt werden oder die Reports auf bestimmte Alarm Typen (Host oder Service Alarme), Statustypen oder Statuscodes beschränkt werden.

Notifications Unter diesem Punkt werden die Benachrichtigungen, welche an Kontakte oder Kontaktgruppen gesendet wurden, aufgelistet.

Event Log Hier kann in den archivierten Nagios-Logs geblättert werden. Dazu muss die *log rotation* in der Nagios Hauptkonfigurationsdatei aktiviert sein (der Parameter *log_rotation_method*). Die Logdateien liegen standardmäßig unter */var/log/nagios3/archives*.

3.4 Verarbeitung von Log-Daten

Das Verb *loggen* (Englisch: to log) stammt vom Nomen *Logbuch* (Englisch: logbook) ab und bezeichnet in der EDV das systematische Aufzeichnen von bestimmten



Service 'PING' On Host 'gateway'

2008-12-07 00:00:00 to 2008-12-07 18:59:09
Duration: 0d 18h 59m 9s

First as
Curren
Report
Today

[Availability report completed in 0 min 0 sec]

Service State Breakdowns:

State	Type / Reason	Time	% Total Time	% Known Time
OK	Unscheduled	0d 18h 59m 9s	100.000%	100.000%
	Scheduled	0d 0h 0m 0s	0.000%	0.000%
	Total	**0d 18h 59m 9s**	**100.000%**	**100.000%**
WARNING	Unscheduled	0d 0h 0m 0s	0.000%	0.000%
	Scheduled	0d 0h 0m 0s	0.000%	0.000%
	Total	**0d 0h 0m 0s**	**0.000%**	**0.000%**
UNKNOWN	Unscheduled	0d 0h 0m 0s	0.000%	0.000%
	Scheduled	0d 0h 0m 0s	0.000%	0.000%
	Total	**0d 0h 0m 0s**	**0.000%**	**0.000%**
CRITICAL	Unscheduled	0d 0h 0m 0s	0.000%	0.000%
	Scheduled	0d 0h 0m 0s	0.000%	0.000%
	Total	**0d 0h 0m 0s**	**0.000%**	**0.000%**
Undetermined	Nagios Not Running	0d 0h 0m 0s	0.000%	
	Insufficient Data	0d 0h 0m 0s	0.000%	
	Total	0d 0h 0m 0s	0.000%	
All	Total	0d 18h 59m 9s	100.000%	100.000%

Service Log Entries:
[View full log entries]

Event Start Time	Event End Time	Event Duration	Event/State Type	Event/State Information
2008-12-06 12:05:45	2008-12-06 12:05:46	0d 0h 0m 1s	SERVICE OK (HARD)	First Service State Assumed (Faked Log Entry)

Abbildung 3.1: Nagios Availability Reporting

Datenverarbeitungs-Ereignissen.[28]

So fallen auch die Performance-Daten, welche von den Nagios Plugins geliefert werden, in diese Kategorie der Log-Daten. Jedoch haben diese Daten einige spezielle Eigenschaften welche es zu beachten gilt. In diesem Abschnitt wird die generelle Problematik beim Speichern von Log-Daten erläutert und im Anschluss werden die Besonderheiten bei Nagios Performance-Daten erläutert und welche Konsequenzen sich daraus möglicherweise ergeben.

3.4.1 Generelle Problematik

Der Grund warum Log-Daten gespeichert werden, ist, dass man diese später bei Bedarf oder auch routingemäßig analysieren möchte. In einer größeren IT-Infrastruktur werden Daten von verschiedensten Anwendungen auf unterschiedlichsten Systemen geschrieben. Im folgenden einige populäre Beispiele für Log-Daten.

- Log-Daten eines Rechners in */var/log/messages* (Linux/Unix) oder in den *Ereignisprotokollen* (Windows)
- Log-Daten von verschiedensten Server Anwendungen wie Mail-Server, Proxy-Server, DNS-Server, DHCP-Server, File-Server, Datenbank-Server etc.
- Log-Daten von Installationsroutinen
- Log-Daten eines Webservers
- Log-Daten des Virenscanners

Diese wenigen Beispiele sind natürlich keine erschöpfende Aufzählung, man könnte die Liste beliebig fortführen, denn prinzipiell kann jede Software ihre eigenen Log-Datei schreiben. Eine Besonderheit stellen die Log-Daten eines Webservers dar. Erstens ist hier der sensible Umgang mit den Daten besonders wichtig, denn oft

[28] vgl. en.wikipedia, 2009-01-14, 1, Data logging, http://en.wikipedia.org/wiki/Data_logging

wird es sich um Daten von Anwendern handeln, welche nicht ohne weiteres langfristig gespeichert werden dürfen und zweitens ist hier die Analyse ein besonderes Thema. Es kann so das *Surf-Verhalten* der User analysiert werden und auf Grund der Ergebnisse zum Beispiel die Benutzerfreundlichkeit verbessert werden. Dies ist aber nur ein Beispiel, wofür die Log-Datei eines Webservers verwendet werden kann. Die Möglichkeiten sind äußerst vielfältig und Gegenstand vieler Abhandlungen. Dieses Thema soll uns hier aber nicht weiter beschäftigen.[29]

Gespeichert werden die Log-Daten meist in, vom Menschen lesbaren, Textdateien. Es ist aber auch schon vermehrt der Fall, dass Daten direkt in relationale Datenbanken geschrieben werden. Durch dieses verstreute Speicherung der Log-Daten von unterschiedlichen Anwendungen in unterschiedliche Datenspeicher ergeben sich mehrere Herausforderungen.

- Zusammenführung aller Log-Daten auf einem zentralem System
- Verarbeitung von unterschiedlichen und teilweise undokumentierten Formaten
- Erkennung von ungültigen Log-Einträgen
- Aufbewahrung von großen Datenmengen über eine lange Zeitdauer

Die Erfüllung dieser Herausforderungen und die anschließenden Analyse der Daten wird unter dem Begriff *Log Management* zusammengefasst.[30]

Die drei erstgenannten Problemstellungen haben für die Nagios Performance-Daten kaum Relevanz. Hier müssen die Log-Daten nicht erst auf ein zentrales System zusammengeführt werden, da Nagios genau dies, als zentraler Netzwerkmonitor, schon macht. Die Problematik der unterschiedlichen Formate tritt in abgeschwächter Form auf. Denn auch die Plugins können verschiedenste Formate für

[29] vgl. wikipedia, 2009-01-15, 7, Logdatei, http://de.wikipedia.org/wiki/Logdatei

[30] vgl. en.wikipedia, 2009-01-14, 2, Log management and intelligence, http://en.wikipedia.org/wiki/Log_management_and_intelligence

die Performance-Daten verwenden oder sogar gänzlich auf eine Performance-Daten Ausgabe verzichten. Um die Verwendbarkeit ihrer Plugins zu optimieren, werden sich jedoch die meisten Plugin-Entwickler an die im Abschnitt 3.1.4 auf Seite 19 beschriebene Struktur halten. Denn prinzipiell wird genau für eine Anwendung (nämlich Nagios) entwickelt. Ungültige Log-Einträge haben eigentlich nur in Form von syntaktisch falsch eine Bedeutung. Da es bei diesen Log-Daten nicht um den Nachweis von sicherheitsrelevanten Daten geht, spielt eine mögliche Manipulation der Daten kaum eine Rolle. Eine syntaktische Prüfung sollte kein Problem darstellen.

Die Problematik enorme Datenmengen bewältigen zu müssen und diese über lange Zeiträume zu speichern, ist jedoch im Bereich der Nagios Performance-Daten ebenso von Bedeutung. Und genau dies ist eine der größten Herausforderungen, denn in größeren Infrastrukturen fallen jährlich bis zu 40 TB an Log-Daten an.[31] Allgemein gesehen, muss man erstens Wege finden, den Speicherbedarf in vernünftigen Grenzen zu halten und zweitens muss man darauf achten, dass die Daten weiterhin performant und sinnvoll zu verarbeiten bzw. analysieren sind.

Ein weiterer Aspekt, welcher bei der Speicherung von Log-Daten eine immer größere Rolle spielt, ist die rechtliche Lage. Also ist man überhaupt berechtigt, die Informationen zu speichern. Dies ist vor allem von Bedeutung, wenn man Kunden- bzw. Anwenderdaten speichert. Ein populäres Beispiel sind IP-Adressen. Speichert man nur Daten aus der eigenen IT-Infrastruktur, also typische Nagios Performance-Daten, so wird diese Problematik kaum eine Rolle spielen. Jedoch sollte man dieses Thema immer im Hinterkopf behalten, da es einem schnell große Unannehmlichkeiten bereiten kann.

Die zentrale Speicherung der Log-Daten wird im Regelfall in relationalen Datenbanken erfolgen, da diese durch ihre Struktur und Flexibilität die Aufbereitung und Auswertung deutlich erleichtern. Mehr dazu, mit Augenmerk auf Nagios Performance-Daten, im Abschnitt 4.3.2 auf Seite 56.

[31] vgl. MacKinnon, 2009-01-14, 1, LMI In The Enterprise,
http://www.processor.com/editorial/article.asp?article=articles%2Fp2746%2F09p46%2F09p46.asp

Die größte Bedeutung ist wohl der Analyse der Log-Daten beizumessen. Denn genau diese Analyse ist ja der Grund, warum die Daten überhaupt aufgezeichnet werden. Bei geringen Datenmengen auf kleinen Systemen (einzelner Rechner), ist es durchaus gebräuchlich die Analyse manuell durchzuführen, indem man die Logdatei durchgeht und auf entsprechende Merkmale hin untersucht. Bei enormen Mengen von Logdaten (man denke nur an 40 TB im Jahr), ist eine manuelle Analyse jedoch nicht mehr möglich und es muss Software eingesetzt werden, die einen großen Teil dieser Arbeit automatisiert. Software, welche zum *Log Management* eingesetzt wird, deckt den ganzen Prozess ab. Also Datensammlung, Datenspeicherung, automatisches Erzeugen von Reports auf Basis der Log-Daten und generieren von Warnmeldungen bei kritischen Ereignissen und Verläufen. Alles in allem ähneln diese Aufgaben sehr dem Verhalten von Nagios und seinen Addons, was wenig verwunderlich ist, denn auch hier handelt sich um Log-Daten, wenn auch eine sehr spezielle Form. Mehr zu den besonderen Eigenschaften im nächsten Abschnitt.[32]

3.4.2 Besonderheiten bei Nagios Performance-Daten

Wie bereits erwähnt haben die Performance-Daten von Nagios, welche generell zu den Log-Daten zu zählen sind, einige spezielle Eigenschaften. Diese werden nun beschrieben.

Log-Daten sind Zahlenwerte

Die Performance-Daten sind, abgesehen von Einheiten, zumeist reine Zahlenwerte. Theoretisch könnten Plugins auch Daten liefern, welche keine Zahlenwerte sind, dies ist jedoch für uns nicht interessant, da diese dann ohnehin nicht in Diagrammen dargestellt werden könnten. Durch diese Besonderheit, dass man es nur mit Zahlenwerten zu tun hat, ergebenn sich gewisse Konsequenzen und Vorteile. Zunächst

[32] vgl. MacKinnon, 2009-01-14, 1, LMI In The Enterprise,
http://www.processor.com/editorial/article.asp?article=articles%2Fp2746%2F09p46%2F09p46.asp

können die Daten, wie erwähnt, leicht grafisch dargestellt werden. Ein weitere Vorteil ist, dass man die Daten leicht zusammenfassen kann. Man kann also Durchschnittswerte, Maxima/Minima oder ähnliches über bestimmte Zeiträume bilden und so die Daten zwar mit geringerer Genauigkeit speichern, jedoch auch enorm Speicherplatz sparen. Dies ist vor allem bei einer langfristigen Datenhaltung interessant. Weiters ist es so, dass RRDtool, die Implementierung einer Round-Robin Datenbank, welche im Verlauf dieser Arbeit noch genauer besprochen wird, nur Zahlenwerte aufnimmt. Es ist also die Nutzung dieser Software möglich.

Besondere Motivation / Verwendungszweck

Die Motivation für die Aufzeichnung ist zumeist die selbe, man möchte die Daten in irgendeiner Form analysieren, um dann entsprechende Schlüsse zu ziehen. Bei herkömmlichen Log-Daten ist es so, dass die Daten an sich analysiert werden. Ob dies jetzt manuell oder automatisiert geschieht, sei dahingestellt. Die Nagios-Performance Daten wird man in den seltesten Fällen direkt einsehen und analysieren. Wenn, dann erst im zweiten Schritt. In erster Linie, wird man die Daten in Diagrammen darstellen und versuchen aus den Diagrammen Informationen abzulesen. Man zeichnet die Daten also auf, um sie dann darzustellen. Daraus werden sich möglicherweise Konsequenzen für die Speicherung ergeben. Denn die Daten sollen so gespeichert und in einem passenden Datenspeicher gehalten werden, damit sie effizient dargestellt werden können. Nicht um sie für den Menschen lesbar aufzubereiten.

Übernahme der Log-Daten

Wie bereits im Abschnitt 3.1.4 auf Seite 19 beschrieben, gibt es verschiedene Schnittstellen um die Performance-Daten der Nagios Plugins entgegenzunehmen. Dabei ist die direkte Verarbeitung über *Befehle* bei einer großen IT-Infrastruktur zu vermeiden, da dies die CPU zunehmend belasten würde. Auch ist bei Performance-Daten

Verarbeitung über eine Datei, die direkte Verarbeitung der Datei, mit Vorsicht zu genießen. Denn hier wird Nagios für die Verarbeitungsdauer blockiert. Ideal ist eine Verarbeitung wie sie *PNP4Nagios* im *Bulk Mode with NPCD* praktiziert wird. So wird die Performance optimiert und die Blockierung von Nagios minimiert.

Kein Nebenprodukt

Eine sehr wichtige Eigenschaft, welche die Nagios Performance-Daten vom Großteil aller anderen Log-Daten unterscheidet, ist, dass es sich hierbei nicht um ein Nebenprodukt handelt. Ein Mail-Server hat z.B. die Hauptaufgaben Emails zu verwalten und zu versenden. Um die Tätigkeit zu dokumentieren, werden Log-Daten erzeugt. Die Log-Daten sind hier ein Nebenprodukt. Bei Nagios ist dies gänzlich anders. Das generieren von Log-Daten (unter anderem Performance-Daten) ist der Zweck, warum Nagios überhaupt eingesetzt wird. Die Log-Daten sind hier also der zentrale Inhalt.

Schnelle Verfügbarkeit

Unter anderem, genau weil die Log-Daten bei Nagios der zentrale Inhalt sind, möchte man diese möglichst schnell zur Verfügung haben. Der Idealfall wäre, würde man sozusagen *Live* Diagramme zu den überwachten Hosts und Services haben. Die Notwendigkeit, dass die Daten möglichst schnell gespeichert und aufbereitet werden unterscheidet die Performance-Daten von herkömmlichen Log-Daten. Daher ist der Performance beim Speichern und Verarbeiten der Daten größte Aufmerksamkeit zu schenken. Mehr dazu im Kapitel 4.4 auf Seite 58.

Trotz all dieser Besonderheiten und Unterschiede, haben herkömmliche Log-Daten und die Nagios Performance-Daten viel gemeinsam. Denn auch die Performance-Daten sind eine Form von Log-Daten, wenn auch eine sehr spezielle. Die für uns bedeutendste Gemeinsamkeit ist, dass beide in hohe Frequenz anfallen und bei

beiden müssen Mittel und Wege gefunden zu werden, um enorme Datenmengen abzulegen und zu verwalten.

Kapitel 4

Vergleich von RRD und relationaler DB

4.1 RRD - RRDtool

RRDtool ist eine konkrete Sofwareimplementierung einer Round Robin Database (RRD) und wird als einzige RRD Implementierung zur Speicherung von Performance Daten im Bereich der Nagios Addons eingesetzt. Eben wegen dieser Sonderstellung und auf Grund der Tatsache, dass RRDtool die deutlich populärste Implementierung einer RRD ist, wird in diesem Kapitel weniger allgemein das Konzept RRD besprochen, sondern das Augenmerk auf RRDtool gelegt.

RRDtool ist eine Software mit der zeitbezogene Daten gespeichert und visualisiert werden können. Das ursprünglich von Tobias Oetiker entwickelte Programm ist mittlerweile als freie Software verfügbar und wurde von verschiedenen Autoren weiterentwickelt. RRDtool liegt aktuell in der Version 1.3.4 vor (letztes Update vom 5. Oktober 2008) und wird laufend weiterentwickelt. Die Bedienung erfolgt über die Kommandozeile bzw. über Programmierschnittstellen, welche für mehrere Programmiersprachen existieren. Etliche Programme machen sich diese Schnittstellen zu Nutze und verwenden RRDtool für ihre Zwecke, so auch ein großer Teil der

Graphing und Trending Addons.[1]

4.1.1 Speicherung und Zusammenfassung der Daten

Die Abkürzung *RRD* steht für *Round-Robin-Database* und gibt schon einen Hinweis auf die Art und Weise, wie RRDtool mit den Daten umgeht. *Round-Robin* ist ein Begriff, den man in der Informatik des öfteren antrifft und beschreibt in diesem Fall, wie die Speicherung der Daten erfolgt.

Eine RRD (oder RRD File) kann mehrere Round Robin Archives (RRA) beinhalten. Die RRD Datei repräsentiert im Regelfall eine bestimmte Datenquelle, deren Daten dann in den RRA gespeichert werden. Es können (und werden meistens) mehrere RRA pro RRD existieren, da pro RRA die Auflösung (also die Genauigkeit der Daten) und die Zeitdauer bestimmt wird. So können z.B. in einem RRA Daten über eine große Zeitspanne (beispielweise ein Jahr) mit eher geringer Auflösung (z.B. Stundendurchschnitt) und in einem weiteren Daten über eine geringere Zeitdauer (beispielsweise eine Woche) mit höherer Auflösung (z.B. jede Minute ein Wert) gespeichert werden.

Wird das Ende der Zeitspanne erreicht, für die Daten in einem RRA gespeichert werden, so wird wieder am Anfang des RRA begonnen die ältesten Daten zu überschreiben. Daher die Bezeichnung Round-Robin. So hat eine RRD mit ihren RRA von Anfang an eine fixe Größe (die RRA werden zu Beginn mit *UNKNOWN* Werten gefüllt) welche sich nicht mehr verändert.[2]

rrdcreate

Mit dem Kommandozeilenaufruf *rrdtool create* wird eine neue RRD angelegt. Das Verständnis einiger Optionen dieses Kommandos ist wichtig, um RRDtool an sich zu verstehen. Die Syntax des Aufrufs sieht folgendermaßen aus:

[1] vgl. wikipedia, 2008-12-08, 4, RRDtool, http://de.wikipedia.org/wiki/Rrdtool

[2] vgl. Oetiker, 2008-12-09, 1, RRDtool - rrdcreate, http://oss.oetiker.ch/rrdtool/doc/rrdcreate.en.html

```
1 rrdtool create filename [--start|-b start time] [--step|-s step] [
    DS:ds-name:DST:dst arguments] [RRA:CF:cf arguments]
```

Wobei *filename* den Dateinamen der RRD Datei spezifiziert. Mittels des *time* Arguments wird (in Sekunden seit 1970-01-01 UTC) angegeben, ab wann der erste Wert akzeptiert wird. Ältere Werte werden nicht in die RRD aufgenommen. Das *step* Argument gibt an, mit welchem Mindestinvervall Daten in die RRD aufgenommen werden. Standardmäßig sind es 300 Sekunden. Was bedeutet, dass alle 300 Sekunden ein neuer Wert für die RRD angenommen wird. Diese Werte werden als *Primary Data Points (PDP)* bezeichnet.

Mittels *DS* (Data Source) wird die Datenquelle näher beschrieben. *ds-name* ist ein frei zu vergebender Name, welcher die Datenquelle treffend bezeichnen sollte. *DST* gibt den *Data Source Type* an. Folgende Typen sind möglich: GAUGE, COUNTER, DERIVE, ABSOLUTE und COMPUTE. So ist *GAUGE* beispielsweise für Messwerte (z.B. Temperatur) zu verwenden oder *COUNTER* für Werte die auf jeden Fall ansteigen (Overflows werden mitgezählt). Eine genaue Beschreibung der *Data Source Types* ist der man-page für *rrdcreate* zu entnehmen. Abgesehen von *COMPUTE* (worauf hier nicht näher eingegangen wird) sind die *dst arguments* folgendermaßen anzugeben:

```
1 DS:ds-name:GAUGE | COUNTER | DERIVE | ABSOLUTE:heartbeat:min:max
```

heartbeat definiert die Zeitspanne (in Sekunden) die zwischen zwei Updates der RRD vergehen darf, bevor ein *UNKNOWN* Wert geschrieben wird. Der Wert ist sinnvollerweise ein vielfaches des *step* Werts. *min/max* beschreibt den zulässigen Bereich der Datenwerte. Ist ein Wert nicht innerhalb dieses Bereichs wird er als *UNKNOWN* angenommen.

Große Bedeutung hat das *RRA* Argument. Hier wird beschrieben, wie die eigentliche Speicherung im *Round-Robin Archive* erfolgt. Dieses Argument kann natürlich mehrmals auftreten. *CF* spezifiziert die *consolidation function* (Konsolidierungs- oder Zusammenführungsfunktion), also wie werden mehrere Werte (Primary Data Points

- PDP) auf einen CDP (Consolidated Data Point) zusammengeführt, wenn die Auflösung der Daten verringert wird. Die möglichen CF sind AVERAGE, MIN, MAX und LAST. Sie sollten selbsterklärend sein. Die *cf arguments* sind folgendermaßen anzugeben:

```
1 RRA:AVERAGE | MIN | MAX | LAST:xff:steps:rows
```

xff beschreibt den zulässigen Anteil von *UNKNOWN* PDPs im Konsolidierungsintervall. Dieser Wert kann zwischen 0 und 1 sein. So meint zum Beispiel ein *xff* Wert von 0.5, dass bei 6 PDPs die zusammengefasst werden maximal 3 UNKNOWN sein dürfen, damit ein zulässiger konsolidierter Wert entsteht. *steps* gibt an, wieviele PDPs mittels der CF zusammengefasst werden. *rows* definiert dann, wieviele so entstande Werte im RRA gespeichert werden. Hiermit wird also die Zeitspanne definiert.

Ein Beispiel. Bei einer Schrittweite von 300 Sekunden, in der Daten in die RRD aufgenommen werden (alle 300 Sekunden ein PDP), einer Schrittweite von 12 (12 PDPs werden zusammengefasst) für die CF *AVERAGE* und 2400 *rows* bedeutet das, dass der Stundendurchschnitt (300 Sekunden mal 12 sind 3600 Sekunden) gespeichert wird und dies über 100 Tage (2400 Werte, also 2400 Stundenmittel - 2400 Stunden durch 24 sind 100 Tage).[3]

rrdupdate

Für die Performance Messungen, welche im Abschnitt 4.4 auf Seite 58 stattfinden, wird *rrdupate* verwendet, um Daten in die RRD zu schreiben. Die vereinfachte, für die hier vorliegenden Zwecke ausreichende Syntax dieses Kommandozeilenbefehls sieht folgendermaßen aus:

```
1 rrdtool update filename N|timestamp:value
```

Wobei *filename* der Name der RRD Datei ist. Dann kann entweder ein Timestamp (in Sekunden seit 1970-01-01 UTC) oder N, was bedeutet, dass der Wert mit aktuellen

[3] vgl. Oetiker, 2008-12-09, 1, RRDtool - rrdcreate, http://oss.oetiker.ch/rrdtool/doc/rrdcreate.en.html

Zeit eingefügt wird, angegeben werden. *value* ist der einzufügende Wert.[4]

4.1.2 Darstellung der Daten

Der Kommandozeilenbefehl *rrdtool graph* dient dazu, die Daten für den Menschen lesbar darzustellen. Vorrangig wird er dazu verwendet Diagramme zu erstellen, es können jedoch auch rein numerische Reports generiert werden.

Die Möglichkeiten, welche *rrdgraph* bietet sind äußert vielfältig und die Handhabung ist sehr flexibel. Es können nicht einfach nur die Daten, welche in den RRA vorhanden sind, direkt in Diagrammen dargestellt werden, sondern zuvor noch diverse Manipulationen an den Daten vorgenommen werden. In groben Zügen sieht die Syntax des *rrdgraph* Aufrufs folgendermaßen aus:

```
1 rrdtool graph filename [option ...] [data definition ...] [data
    calculation ...] [variable definition ...] [graph element ...] [
    print element ...]
```

Wobei *filename* der Dateiname des Outputfiles ist. Für die Ausgabe können verschiedene Formate (z.B. PNG, SVG, PDF etc.) definiert werden. Dies und etliche weitere Einstellungen können durch die Optionen (*option*) festgelegt werden.

Im nächsten Schritt wird mit dem *data definition*, *data calculation* und/oder dem *variable definition* Kommando die Auswahl der Daten für die Darstellung getroffen. Für das *data calculation* und das *variable definition* Kommando wird eine eigene Sprache verwendet, welche *RPN* genannt wird. Die Verwendung von *RPN* wird in der man-page *rrdgraph_rpn* beschrieben. Die Aufbereitung der Daten, welche in der man-page *rrdgraph_data* näher beschrieben ist, bietet diverse Möglichkeiten. So können auch Daten von mehreren RRD Dateien genommen, Berechnungen durchgeführt, Werte zur weiteren Bearbeitung in Variablen abgespeichert werden und vieles mehr.

Die genaue Syntax für das *graph* bzw. das *print* Element ist ebenfalls einer eigenen man-page zu entnehmen (*rrdgraph_graph*). Wie bereits erwähnt, können sowohl Dia-

[4] vgl. Oetiker, 2008-12-09, 2, RRDtool - rrdupdate, http://oss.oetiker.ch/rrdtool/doc/rrdupdate.en.html

gramme gezeichnet als auch rein numerische Reports generiert werden. Für beide Zwecke greift man auf die Daten Definitionen zurück, welche zuvor erzeugt wurden. Also die mit *data definition*, *data calculation* und *variable definition* aufbereiteten Daten, sind über definierte Variablennamen ansprechbar, um nun die eigentliche Ausgabe zu erzeugen.[5]

4.1.3 Weitere Kommandos

Neben den besprochenen Kommandos *rrdtool create*, *rrdtool update* und *rrdtool graph* gibt es noch einige weitere Befehle um die RRD und deren Daten zu verwalten bzw. diese auszulesen.

rrdcgi Der Zweck dieses Kommandos ist, dass es als CGI Skript läuft und dabei ein Template einer Web Seite hinsichtlich spezieller *RRD* Tags parst. Als Endresultat wird eine Web Seite mit den notwendigen CGI Headern ausgegeben.

rrdtool dump Gibt die Inhalte eines RRD Files im XML Format aus.

rrdtool fetch Daten können in gewünschter Auflösung und für ein bestimmtes Zeitfenster ausgegeben werden. Wird vor allem intern von *rrdtool graph* verwendet.

rrdtool first Gibt den UNIX Timestamp des ersten Dateneintrags in ein bestimmtes RRA zurück. Angegeben werden muss das RRD File und der Index des RRA. Der Index kann mit *rrdtool info* bestimmt werden.

rrdtool last Analog zu *rrdtool first* wird hier der letzte Dateneintrag ausgegeben.

rrdtool info Gibt die Header Informationen einer RRD aus.

rrdtool lastupdate Gibt den UNIX Timestamp und den Wert für jedes Datum im letzten Update der RRD aus.

[5] vgl. Oetiker, 2008-12-10, 3, RRDtool - rrdgraph, http://oss.oetiker.ch/rrdtool/doc/rrdgraph.en.html

rrdtool resize Verändert die Größe eines RRA. Auch hier ist RRD File und der Index des RRA anzugeben sowie die Anzahl der Reihen die hinzuzufügen bzw. zu entfernen sind.

rrdtool restore Aus dem XML Dump (mit *rrdtool dump* erzeugt) wieder eine RRD anlegen.

rrdtool tune Hiermit können Konfigurationseinstellungen im RRD Header verändert werden.

rrdtool xport Daten im XML Format exportieren. Im Gegensatz zu *rrdtool dump* können die Daten, welche im XML Format exportiert werden sollen, auf vielfältige Weise ausgewählt werden. Zudem können auch mehrere RRD Files kombiniert werden.[6]

4.1.4 Physikalische Struktur

In diesem Abschnitt wird die Speicherstruktur und der Vorgang beim Speichern in die RRD Files beschrieben. Dieses Hintergrundwissen ist notwendig, um bei den späteren Performance Messungen eventuelle Rückschlüsse auf die Speicherstruktur ziehen zu können.

Von Interesse sind vor allem die Vorgänge bei *rrdtool update* (siehe Abschnitt 4.1.1 auf Seite 40), denn mit diesem Kommando wird in das RRD File geschrieben.

Strukturen

Die verwendeten Strukturen sind in *rrd_format.h* definiert. Grundsätzlich gibt es eine *struct* vom Typ *rrd_t* in welcher alle anderen Strukturen zusammengefasst sind. Diese Strukturen werden nun kurz beschrieben.

stat_head_t In diesem statischen Header ist unter anderem festgehalten wieviele DS (Data Sources) und RRA die RRD beinhaltet und wie die step size der RRD ist.

[6] vgl. Oetiker, 2008-12-22, 4, RRDtool Documentation, http://oss.oetiker.ch/rrdtool/doc/index.en.html

ds_def_t Name und Typ der Data Sources.

rra_def_t Die Einstellungen, welche das RRA betreffen sind hier festgehalten. Also die Consolidation Function (CF), wieviele CDP (Consolidated Data Point) gespeichert und wieviele PDP (Primary Data Point) zu einem CDP zusammengefasst werden.

live_head_t Wann die RRD zuletzt aktualisiert wurde (Genauigkeit in Mikrosekunden)

pdp_prep_t Hier werden die PDP gehalten.

cdp_prep_t In diesem Bereich werden die Daten für einen CDP aufbereitet.

rra_ptr_t Pro RRA in der RRD ist hier ein Pointer festgehalten, welcher auf die aktuelle Zeile des RRA verweist. In einer Zeile stehen Werte (CDP) mit dem selben Alter, also die Werte verschiedener DS. Wobei eine Spalte im RRA die CDP einer einzigen DS darstellt.

Weiters ist noch ein Pointer auf die Werte (rrd_value) in *rrd_t* enthalten. Die Werte sind vom Typ *rrd_value_t* - in *rrd.h* als herkömmlicher *double* definiert.

Prinzipiell ist es also so, dass sich RRDtool für jedes RRA die aktuelle Zeile merkt, es muss also beim Eintragen in die RRD nicht gesucht werden, da beim Schreiben einfach sequentiell vorgegangen wird. Dies entspricht dem Round-Robin Verhalten und sollte auch sehr performant sein. Beim Abrufen der Daten, z.B. für bestimmte Zeiträume, muss jedoch nicht sequentiell gesucht werden, da die Intervalle bekannt und gleichbleibend sind und so einfach der Pointer auf die entsprechende Zeile im RRA berechnet werden kann (wahlfreier Zugriff) und ab dort sequentiell gelesen wird.

Ablauf bei Update

Im vorliegenden Absatz werden grob die internen Vorgänge erläutert, welche durch ein *rrdtool update* ausgelöst werden. Ausgangspunkt ist *rrdupdate.c*, hier wird im *main* eigentlich nur die Funktion *rrd_update* aufgerufen, welche in der Datei *rrd_update.c* ausprogrammiert ist. Auch in dieser Funktion passiert noch nicht viel Erwähnenswertes. Hauptsächlich werden die Optionen und Argumente, welche beim Aufruf übergeben wurden, überprüft. Abschließend wird die Funktion *rrd_update_r* aufgerufen, welche selbst wiederum die Funktion *_rrd_update* aufruft (beide im selben Source File zu finden). In *_rrd_update* wird mit *rrd_open* (zu finden in *rrd_open.c*) das angegebene RRD File geöffnet, eine *rrd_t* Struktur initialisiert und für die übergebenen Argumente *process_arg* aufgerufen. Die Werte werden hier und in weiteren Funktionen verarbeitet und zuletzt in das RRD File selbst geschrieben (*rrd_write* in *rrd_open.c*).

Weitere Aspekte

Grundsätzlich ist noch anzumerken, dass im Regelfall für jeden Host- oder Service Check eine RRD, also ein RRD File, existieren wird. Man darf beim Eintragen der Performance-Daten also nicht nur an die Vorgänge denken, welche RRDtool selbst erledigt, sondern muss auch bedenken, dass das Dateisystem erst die entsprechende Datei (zum Check passende RRD File) auffinden muss. Gibt es sehr viele und auch schon entsprechend große RRD Files, so können diese auch schon weit verteilt auf der Festplatte liegen. Da die Performance-Daten der Checks nicht immer in der selben Reihenfolge geliefert werden, werden die Dateien auch nicht immer in der selben Reihenfolge aufgerufen. So kann es durchaus zu entsprechenden Verzögerungen kommen, wenn in unterschiedlichster Reihenfolge weit auseinanderliegende Regionen der Festplatte adressiert werden müssen.

4.2 Relationale Datenbank

Grundsätzlich wird davon ausgegangen, dass sich der Leser im Klaren über den Begriff *relationale Datenbank* ist. Dieser Abschnitt bietet keine detaillierte Einführung in die Welt des relationalen Datenbankmodells, der Normalformen und ähnlichem, sondern soll nur, der Vollständigkeit halber und um Missverständnissen vorzubeugen, über die Charakteristika einer relationalen Datenbank Auskunft geben.

Eine relationale Datenbank dient, wie wohl auch jede andere Datenbank auf einem Computersystem, zur elektronischen Datenverwaltung. Zur Definition, Abfrage und Manipulation von Daten wird überwiegend die Datenbanksprache *SQL* eingesetzt. Grundlage und Namensgebend für das Konzept der relationalen Datenbank ist der mathematische Begriff der *Relation*. Die Datenbank baut auf dem relationalen Datenbankmodell auf und Operationen werden durch die relationale Algebra bestimmt. Auf die mathematischen Hintergründe soll hier nicht näher eingegangen werden, im folgenden wird nur die Struktur einer relationalen Datenbank kurz erläutert und die Operationen werden vorgestellt.

4.2.1 Struktur

Eine relationale Datenbank besteht aus einer Reihe von *Tabellen* (repräsentieren die Relationen). Jede Zeile (Tupel) einer Tabelle stellt einen Datensatz dar. Wobei jedes Tupel aus einer Reihe von Attributwerten besteht. Die *Attribute* sind also die Spalten der Tabelle. Ein Datensatz ist nicht über die Position in der Tabelle ansprechbar (die Reihenfolge ist nicht definiert), sondern muss ein Merkmal zur eindeutigen Identifikation besitzen. Den sogenannten *Schlüssel* (engl. Key). Dieser Schlüssel kann ein bestimmtes Attribut sein oder sich auch aus mehreren Attributen zusammensetzen (zusammengesetzter Schlüssel). Der Schlüssel muss für jeden Datensatz eindeutig sein.

Die Tabellen können weiters in *Beziehung* zueinander stehen. So können ein oder

mehrere Attribute in einer Tabelle einen sogenannten *Fremdschlüssel* (engl. Foreign Key) repräsentieren. Dieser ist kein eindeutiger Schlüssel in der Tabelle, in der er Fremdschlüssel ist, er ist jedoch eine Verknüpfung zu einer weiteren Tabelle, in der genau dieses Attribut ein Schlüssel ist.

4.2.2 Operationen

Die *relationale Algebra* beschreibt wie Daten gespeichert, abgefragt und manipuliert werden können. Es gibt einen Satz von Operationen, aus welchen alle anderen möglichen Operationen abgeleitet werden können. Da diese Operationen eine wichtige Charakteristik einer relationalen Datenbank darstellen, werden sie im folgenden kurz vorgestellt.

Projektion Wählt einzelne Attribute aus. Also Auswahl auf Spaltenebene.

Selektion Wählt einzelne Tupel aus. Es werden also Zeilen ausgeblendet.

Kreuzprodukt Alle Kombinationen der Tupel aus mehreren Tabellen. Es wird also jede Zeile einer Tabelle mit jeder Zeile einer anderen Tabelle kombiniert. In Kombination mit der Selektion sind die verschiedensten *Joins* möglich.

Umbenennung Durch diese Operation können Tabellen (Relationen) und Spalten (Attribute) umbenannt werden.

Vereinigung Zusammenführung aller Tupel mehrerer Relationen zu einer einzigen Relation. Nur möglich, wenn die Relationen das gleiche Schema besitzen (gleichen Attribute und Attributtypen).

Differenz Es werden aus einer Relation alle Tupel entfernt, die auch in der anderen Relation vorkommen.[7]

[7] vgl. wikipedia, 2008-12-10, 5, Relationale Datenbank, http://de.wikipedia.org/wiki/Relationale_Datenbank

4.2.3 Storage-Engine

Ein Datenbanksystem besteht generell aus zwei Teilen. Erstens dem Datenbankmanagementsystem (DBMS) (die Verwaltungssoftware) und der eigentlichen Datenbank, also die Menge der zu verwaltenden Daten. Wie die Daten abgelegt werden, hängt von der Storage-Engine ab. Manche Datenbanksysteme (wie z.B. MySQL) bieten einem mehrere Storage-Engines für verschiedene Zwecke. Diese Storage-Engines können dann z.B. hinsichtlich Performance oder Sicherheit optimiert sein.[8]

4.2.4 Implementierungen

Es gibt verschiedenste Implementierungen, welche auf dem Konzept der relationalen Datenbank basieren. Für die Performance Messungen, welche im Abschnitt 4.4 auf Seite 58 durchgeführt werden, werden die Datenbankmanagementsystem PostgreSQL und MySQL verwendet.[9]

4.2.5 Physikalische Struktur

Ebenso wie bei RRDtool soll auch für die relationalen Datenbanksysteme die Speicherstruktur erläutert werden. Da hier jedoch ein Konzept behandelt wird, welches unterschiedliche Implementierungen kennt, sind die Erläuterungen deutlich allgemeiner gehalten.

Tablespace

Prinzipiell spricht man vom *Tablespace*, wenn man den Bereich meint, in dem die Tabellen und deren Daten gespeichert werden. Wie dieser Tablespace organisiert ist, hängt von der *Storage-Engine* der Datenbank ab. Generell unterscheidet man zwischen einem *System Managed Storage (SMS)* und einem *Database Managed Storage*

[8] vgl. wikipedia, 2008-12-17, 6, Datenbank, http://de.wikipedia.org/wiki/Datenbank

[9] vgl. wikipedia, 2008-12-10, 5, Relationale Datenbank, http://de.wikipedia.org/wiki/Relationale_Datenbank

(DMS). Während bei einem SMS die Speicherverwaltung über Betriebssystemfunktionen durchgeführt wird, erledigt dies das Datenbankmanagementsystem bei einem DMS selbst. Beispiel für ein SMS ist *MyISAM* und ein Beispiel für ein DMS ist die die Storage-Engine *InnoDB*. Beide Storage-Engines können beim Datenbanksystem MySQL eingesetzt werden, wobei MyISAM die Standard-Storage-Engine ist.

Allgemein unterscheidet man 4 Arten von Tablespaces. Für uns von Interesse ist vor allem der reguläre Tablespace. In diesem werden die Tabellen und eventuell dazugehörige Indizes abgelegt. Weiters gibt es noch den temporären Tablespace, den LARGE Tablespace (für große Objekte) und den Tablespace in welchem der Systemkatalog abgelegt ist. Im Systemkatalog sind, wie der Name schon erahnen lässt, die Systemtabellen festgehalten. So findet man dort z.B. die Datentypen oder auch eine Tabelle, in welcher die Tabellen vermerkt sind.[10]

Indexstrukturen

Neben der Tabellenstruktur und den Tabellendaten sind im regulären Tablespace also auch noch Indexstrukturen abgelegt. Die Motivation für die Verwendung von solchen Indexstrukturen ist die Beschleunigung von Such- und Sortiervorgängen. Welche Spalten einer Tabelle indiziert werden sollen, ist meist frei wählbar. Einzig der Primärschlüssel wird im Regelfall immer indiziert sein. Ohne Index müsste bei vielen Operationen die ganze Spalte sequentiell durchsucht werden, durch die Indizes, welche meist in einem B^+-Baum abgelegt werden, ist ein schneller Zugriff möglich. Ein B-Baum ist ein balancierter Baum, welcher, im Gegensatz zu einem Binärbaum, in einem Knoten eine Vielzahl von Verweisen auf Kindknoten enthalten kann. Der B^+-Baum ist eine Sonderform des B-Baums. Bei diesem werden die Datenelemente nur in den Blattknoten gespeichert (die inneren Knoten enthalten ausschließlich Schlüssel). Die Baumhöhe bei einem B^+-Baum ist idealerweise gering, so, dass der Zugriff auf die Datenelemente optimiert wird. Wesentlicher Vorteil dieser Form eines

[10] vgl. wikipedia, 2009-01-25, 8, Tablespace, http://de.wikipedia.org/wiki/Tablespace

Suchbaums ist, dass Sekundärindizes eingesetzt werden können. Also ein weiterer, nach einem anderen Ordnungskriterium sortiertert Suchbaum, basiert auf den selben Daten.[11][12]

MyISAM und InnoDB

MyISAM ist die Standard-Storage-Engine für das MySQL Datenbanksystem. MyISAM basiert auf ISAM, welches in früheren Versionen von MySQL verwendet wurde, aber aktuelle (MySQL 5.1) nicht mehr unterstützt wird. Welche Storage-Engine verwendet wird, kann bei MySQL für jede einzelne Tabelle gewählt werden. Wird eine Tabelle mit der MyISAM Engine abgelegt, so werden für diese eine Tabelle drei Dateien angelegt. Erkennbar sind diese Dateien an ihren Endungen und erfüllen folgende Zwecke:

.frm Diese Datei speichert das Format, also die Struktur der Tabelle.

.myd In dieser Datei werden die Daten der Tabelle abgelegt.

.myi Hier werden die zur Tabelle gehörigen Indexstrukturen gespeichert.

Der Dateiname entspricht jeweils dem Tabellennamen. Per Default-Einstellung können MyISAM Dateien maximal 64 Indizes haben, dies lässt sich jedoch durch ein Recompilieren ändern und ist so auf maximal 128 Indizes ausweitbar.[13]

InnoDB ist die, im Gegensatz zu MyISAM, transaktionssichere Storage-Engine von MySQL. Im Gegensatz zu MyISAM verwendet InnoDB eine oder bei Bedarf, wenn z.B. mit einer Datei die maximal Dateigröße des Betriebssystem überschritten werden würde, auch mehrere Dateien, in welchen die Tabellendaten abgelegt werden. Es wird also nicht pro Tabelle eine Datendatei verwaltet. Weiters existieren noch Logfiles, um die Datenbank nach einem unerwarteten Stillstand wieder schnell

[11] vgl. wikipedia, 2009-01-25, 9, Datenbankindex, http://de.wikipedia.org/wiki/Datenbankindex

[12] vgl. wikipedia, 2009-01-25, 10, B+-Baum, http://de.wikipedia.org/wiki/B%2B-Baum

[13] vgl. MySQL Developer Zone, 2009-01-26, 2, MySQL 5.1 Referenzhandbuch :: 14.1, http://dev.mysql.com/doc/refman/5.1/de/myisam-storage-engine.html

in Betrieb nehmen zu können. Bei MyISAM muss die gesamte Datenbank überprüft werden und die Zeit nimmt, im Gegensatz zu InnoDB, mit der Größe der Datenbank zu.

Es ist auch mit InnoDB möglich, ähnlich dem Verhalten von MyISAM, für jede Tabelle eine eigene Datei zu verwenden (Multi-Tablespaces). Dies muss in der Konfigurationsdatei aktiviert werden. Im Gegensatz zu MyISAM werden jedoch auch dann Indizes und Daten in einer Datei pro Tabelle gehalten (und nicht in .myd und .myi aufgesplittet). Die .frm Datei existiert jedoch auch dann, unabhängig von den Einstellungen in der Konfigurationsdatei, für jede Tabelle seperat.

Bei Verwendung von einem *Shared Tablespace* für die Datendateien, ist es mit InnoDB als Storage-Engine möglich, diesen Shared Tablespace auf einem *Raw Device* abzulegen. So können ungepufferte E/A-Zugriffe implementiert und der Dateisystem-Overhead somit vermieden werden.[14]

Eine weitere interessante Eigenschaft der InnoDB Storage-Engine ist, dass die Records einer Tabelle in Primärschlüssel Reihenfolge abgelegt werden. Im Gegensatz dazu, speichert MyISAM die Records in der Reihenfolge des Einfügens. Beide Konzepte haben ihre Vor- und Nachteile. So kann, bei der Verwendung von InnoDB, bei Abfragen, welche den Primärschlüssel als Kriterium verwenden, ein Performance Vorteil erzielt werden. Hingegen kann sich beim Einfügen von Daten in einer Reihenfolge, die sich stark von der Reihenfolge ihrer Primärschlüssel unterscheidet, durchaus ein gewisser Performance Nachteil bemerkbar machen, da die Records erst geordnet werden müssen.[15]

Prinzipiell sollen diese Erläuterungen der Features und Strukturen nicht auf eine spezifische Storage-Engine abzielen. Es soll damit exemplarisch gezeigt werden, wie bei Datenbanken allgemein vorgegangen wird und welche Konzepte möglich

[14] vgl. MySQL Developer Zone, 2009-01-26, 3, MySQL 5.1 Referenzhandbuch :: 14.2, http://dev.mysql.com/doc/refman/5.1/de/innodb.html

[15] vgl. en.wikipedia, 2009-01-27, 3, MyISAM, http://en.wikipedia.org/wiki/MyISAM

sind.

4.3 Featurevergleich

4.3.1 Anforderungen an die Datenbank

Im folgenden werden die Anforderungen an eine Datenbank vorgestellt, welche für die Zwecke der Speicherung von Performance-Daten entstehen.

- Daten schnell in Datenbank ablegen.
- Daten für verschiedene Zeiträume mit unterschiedlichen Auflösungen speichern.
- Daten eventuell über unbestimmt lange Zeiträume speichern.
- Umgang mit eventuell hohem Speicherbedarf.
- Umfangreiche Möglichkeiten um Daten aus DB graphisch darzustellen.
- Daten (Werte) nach beliebigen Gesichtspunkten gefiltert auslesen.
- Flexibilität der Datenstruktur beim Speichern
- Flexible Verwendung der gespeicherten Daten

4.3.2 Erfüllung der Anforderungen

Abgesehen von der Performance beim Ablegen der Daten, die im nächten Abschnitt (Abschnitt 4.4 auf Seite 58) untersucht wird, wird nun die Erfüllung der einzelnen Anforderungen durch eine relationale DB bzw. eine RRD (im speziellen RRDtool) überprüft. Die Tatsache, dass hier eigentlich eine Technologie (relationale Datenbank) mit einem speziellen Produkt verglichen wird, mag auf den ersten Blick etwas ungewöhnlich erscheinen. Diese Maßnahme ist jedoch auf die Sonderstellung von

RRDtool zurückzuführen (siehe Abschnitt 4.1 auf Seite 37) und daher durchaus gerechtfertig.

Daten für verschiedene Zeiträume mit unterschiedlichen Auflösungen speichern

Um die Daten auch über lange Zeiträume (ein Jahr und länger) halten zu können, ohne, dass der notwendige Speicherplatz explodiert, müssen sie ausgedünnt werden. Die ermittelten Daten der letzten Woche beispielsweise oder gar nur der vergangenen 24 Stunden, will man meist jedoch mit höchstmöglicher Genauigkeit aufbewahren. Man sieht, die Daten müssen also in unterschiedlichen Auflösungen abgelegt werden.

Dies ist eine der großen Stärken der RRD, denn wie bereits im Abschnitt 4.1.1 auf Seite 38 beschrieben, wird mit den RRA dieses Ziel erreicht, ohne, dass man sich selbst weiter darum kümmern muss. Das Ausdünnen der Werte, wird, wie im Abschnitt 4.1.1 auf Seite 39 im Detail erläutert, durch RRDtool selbst, entsprechend den Einstellungen der RRA, erledigt. Eine relationale Datenbank bietet diese Möglichkeit von sich aus jedoch nicht. Will man auch bei einer relationalen DB ein ähnliches Verhalten, so muss man erst selbst eine Lösung dafür entwickeln. Es wird jedoch einen gewissen Aufwand bedeuten diese Funktionen (Konsoldierung der Werte und das Ablegen der unterschiedlichen Auflösungen für verschiedene Zeiträume), unter Beachtung der Performance, zu realisieren.

Daten eventuell über unbestimmt lange Zeiträume speichern

Es wird auch Fälle geben, in denen man die Daten unbestimmt lange aufbewahren will. Man weiß also zu Beginn der Aufzeichnungen noch nicht, wann man auf diese Daten verzichten kann.

Dies stellt die RRD vor ein gewisses Problem, da durch das in Abschnitt 4.1.1 auf Seite 38 beschriebene Round-Robin Verhalten die Daten nach einer zuvor festgelegten Zeit auf jeden Fall überschrieben werden. In der heutigen Zeit ist Spei-

cherplatz jedoch relativ günstig zu haben, daher wäre es oft wünschenswert, bei den unterschiedlichsten Auflösungen, nicht zu Beginn das maximale Alter der Daten festzulegen, sondern die Daten unbestimmt lange Zeit aufbewahren zu können. Die relationale DB hat dieses Round-Robin Verhalten natürlich nicht und es können nahezu beliebig oft beliebig viele Daten eingetragen werden. Der Vorteil liegt in diesem Fall also bei der relationalen Datenbank. Der Grund, warum man diesem Punkt nicht allzuviel Bedeutung beimessen sollte, ist, dass mit *rrdtool resize* im Bedarfsfall die Größe der RRD im Nachhinein geändert werden kann. Sind die Daten jedoch schon überschrieben, so ist es dafür zu spät.

Umgang mit eventuell hohem Speicherbedarf

Bei großen zu überwachenden Infrastrukturen ist es nicht unüblich, dass an die tausend oder mehr Hosts und damit einige tausende Services überwacht werden. Wenn man sich dann vorstellt, dass die Services und Hosts teilweise möglicherweise im Minutentakt überwacht werden, kann man sich vorstellen, dass nach einer gewissen Zeit sehr große Datenmengen zustande kommen.

Die RRD bewältigt dies einerseits mit ihrem Round-Robin Feature und andererseits werden die Daten recht sparsam abgelegt. So benötigt eine RRD mit neun RRA und jeweils 525600 Slots nur 37 MB Speicherplatz (angenommen in jedes RRA werden Minutenwerte eingetragen, so werden die Daten erst nach einem Jahr überschrieben. Ein Jahr hat ca. 525600 Minuten). In einer relationalen DB gibt es wie besprochen kein Round-Robin Feature und der Overhead beim Speichern der Daten ist auch deutlich höher. So muss man mit deutlich größeren Anforderungen an die Speicherkapazitäten rechnen. Das Datenbanksystem (also DBMS und DB) wird damit problemlos umgehen können, denn sie sind darauf ausgelegt große Datenmengen zu speichern.

Bei geringen verfügbaren Speicherkapazitäten ist das Verhalten der RRD also durchaus als vorteilhaft zu betrachten. Man sollte diesem Punkt jedoch nicht allzu-

viel Beachtung schenken, da Speicherplatz relativ günstig zu erwerben ist. Wichtiger ist zu beachten, wie sich die Performance bei größeren Datenmengen verändert. Welches Konzept dann besser geeignet ist, ist Gegenstand der Betrachtungen im Kapitel *Performance Messungen*.

Umfangreiche Möglichkeiten um Daten aus DB graphisch darzustellen

Die Addons zeichnen die Datenwerte eben deshalb auf und speichern sie ab, um sie bei Bedarf grafisch darstellen zu können. Daher ist die Bedeutung dieser Anforderung nicht zu unterschätzen.

RRDtool bietet mit *rrdtool graph* wie im Abschnitt 4.1.2 auf Seite 41 beschrieben, umfangreiche Möglichkeiten um die Daten darzustellen. Sollten diese Möglichkeiten nicht ausreichen, so kann man noch immer mit *rrdtool fetch* oder *rrdtool xport* und *rrdtool dump* die Daten selbst auslesen oder sie in einer XML Struktur exportieren und dann beliebig weiterverarbeiten und darstellen. Man ist also nicht auf die Darstellungsmöglichkeiten von *rrdtool graph* beschränkt, sondern kann diese auch selbst erweitern. Die relationale Datenbank bietet selbstverständlich keine integrierten Darstellungsmöglichkeiten, man muss also selbst eine Lösung entwickeln oder auf bereits verfügbare Software zurückgreifen.

Es bieten also beide Ansätze alle Möglichkeiten, RRDtool macht es einem jedoch deutlich einfacher und stellt ein mächtiges Werkzeug zur Diagrammerstellung zur Verfügung. Einschränkend muss man aber hier auch sagen, dass eben die konkrete Softwarelösung RRDtool besprochen wird, jedoch nur allgemein das Konzept relationale Datenbank. Jedoch es ist auch keine konkrete Implementierung eines geläufigen Datenbanksystem bekannt, welche eine integrierte Diagrammerstellung bietet. RRDtool ist eben genau für diese Zwecke entwickelt worden.

Daten (Werte) nach beliebigen Gesichtspunkten gefiltert auslesen

Oft wird man nur bestimmte Werte auslesen wollen, um diese dann einzusehen oder auch um sie darzustellen. Diese Filterung kann zum Beispiel den Zeitraum betreffen, es kann sich jedoch auch um vielfältigere Kriterien handeln. Der Kreativität sind hier keine Grenzen gesetzt.

RRDtool bietet mit *rrdtool xport* (siehe Abschnitt 4.1.3 auf Seite 43) eine gute Möglichkeit um die Daten zu filtern und dann zu exportieren. Jedoch reicht dieses Feature, trotz der unbestritten vorhandenen Qualität, nicht an die Möglichkeiten von SQL, der Datenbanksprache der relationalen Datenbanken, heran. SQL ist seit mittlerweile über 20 Jahren dahingehend optimiert, nach allen erdenklichen Möglichkeiten Projektionen und Selektionen (siehe Abschnitt 4.2.2 auf Seite 47) vorzunehmen. Dieser Punkt geht damit an die relationalen Datenbanken auch wenn RRDtool für viele Zwecke ausreichende Möglichkeiten bietet. Aber man muss auch beachten, dass viele Menschen mit der Syntax und der Handhabung von SQL gut vertraut sind und will man mit RRDtool ähnliches erreichen, so muss man sich erst damit befassen.

Flexibilität der Datenstruktur beim Speichern

Vorrangig sollen natürlich die ermittelten Werte, also die Performance-Daten, in der Datenbank gespeichert werden. Es wird möglicherweise aber auch die Notwendigkeit bestehen, weitere Daten, welche mit den ermittelten Werten in Zusammenhang stehen, abzulegen. Dies können Informationen sein, welche ebenfalls in der Performance-Daten Struktur, entsprechend den *Nagios plugin-in development guidlines*, enthalten sind. Also Schwellenwerte, Grenzwerte (Minima und Maxima) und die Einheit der Daten. Aber auch Daten welche, abhängig vom Addon, weit darüber hinausgehen. Hier wäre dann also eine absolute Flexibilität der Datenstruktur gefordert.

RRDtool ist eigentlich dafür gedacht, nur die tatsächlichen Zahlenwerte zu spei-

chern. Die Speicherung von weiteren Daten ist bei diesem Produkt nicht vorgesehen. Nur mit einer relationalen Datenbank ist größtmögliche Flexibilität zu erreichen. Diese bieten vielfältige Möglichkeiten, was die Struktur der zu speichernden Daten angeht. In vielen Fällen werden die Möglichkeiten, welche RRDtool bietet, ausreichend sein, hat man jedoch eine höhere Anforderungen an die Datenstruktur, so wird man entweder zur relationalen Datenbank als Speichermöglichkeit für die Performance Daten greifen oder man muss zusätzliche zu den RRD Dateien eine relationale Datenbank betreiben, um den Anforderungen zu genügen. Ob es empfehlenswert ist, sowohl eine RRD als auch eine relationale Datenbank zu betreiben, ist situationsabhängig.

Flexible Verwendung der gespeicherten Daten

In vielen Fällen wird man die Datenbank nicht nur für einen Zweck, also für das Addon, für welches die Daten eigentlich aufgezeichnet werden, verwenden wollen. Hat man die Daten schon abgelegt, so ist es naheliegend, sie auch anderen Anwendungen zur Verfügung zu stellen. Eine Speicherung der Daten pro Anwendung wäre unnötiger Overhead und ist, wenn möglich, zu vermeiden.

Dies ist einer der großen Pluspunkte der relationalen Datenbank. Denn nahezu alle Programmiersprachen bieten vordefinierte Schnittstellen um mit der relationalen Datenbank zu kommunizieren. SQL ist *die* Datenbanksprache und für die verschiedenen Datenbanksysteme nahezu identisch. So ist die Anbindung an die Datenbank in verschiedensten Applikationen sehr leicht und elegant umsetzbar und die meisten Softwareentwickler wissen genau wie sie mit der Datenbank umzugehen haben. Auch sind die relationalen Datenbanken für den Zugriff über das Netzwerk oder Internet optimiert.

RRDtool ist klarerweise längst nicht so verbreitet und daher ist auch die Kompatibilität bei weitem nicht so gegeben. Schnittstellen werden nur zu wenigen Programmiersprachen geboten, der Zugriff über Netzwerk ist nur über Umwege möglich

und das Wissen für den Umgang mit RRDtool weit weniger verbreitet. Die flexible Verwendung der Daten ist jedoch ein sehr wichtiger Punkt und die Erfüllung dieser Anforderung ist nur durch eine relationale Datenbank zu erreichen.

4.4 Performance Messungen

4.4.1 Einleitung

In diesem Abschnitt wird die Performance bei der Speicherung von Daten in unterschiedliche Datenbanken gemessen. Untersucht wird die Performance beim Schreiben in RRD Files und beim Schreiben in eine MySQL bzw. eine PostgreSQL Datenbank. Im folgenden die Eckdaten zum Testsetup.

- Debian *lenny* mit einem Kernel der Version 2.6.26
- RRDtool 1.3.1
- MySQL Server 5.0.51
- PostgreSQL Server 8.3.5

Um die Performance der Datenbanken zu messen, wird das *time* Kommando eingesetzt. Dieses Kommando sollte zwar auf jedem UNIX/Linux System vorhanden sein, jedoch gestalten sich die vorhandenen Optionen nicht immer gleich. Daher kurz eine Erklärung von *time* und der verwendeten Option.

Das Kommando *time* führt ein Programm aus und fasst dabei die Nutzung der System Ressourcen zusammen. Mit der Option *-f FORMAT* bzw. *–format FORMAT* kann ein Formatstring angegeben werden. Der Formatstring besteht für gewöhnlich aus einem oder mehreren *Resource Specifiers*. Wird kein Formatstring angegeben, so kommt ein Standardformat zur Anwendung. Der im vorliegenden Fall verwendete *Resource Specifier e* beschränkt das *time* Kommando auf die Ausgabe der vom auf-

gerufenen Programm benötigte Gesamtzeit. Die Ausgabe erfolgt dann in Sekunden, mit einer Genauigkeit auf 2 Nachkommastellen.

Die Syntax vom *time* Kommando sieht also in unserem Fall folgendermaßen aus:

```
time [ -f FORMAT ] COMMAND
```

4.4.2 Datenstruktur

Für die Struktur, wie die Daten in der RRD bzw. den relationalen Datenbanken abgelegt werden, wurden Anleihen von *pnp4nagios* genommen. Nach der Installation von *pnp4nagios* (siehe Abschnitt 3.2.1 auf Seite 26) ist in der Datei */usr/local/nagios/etc/pnp/rra.cfg-sample* die Erzeugung der RRA dokumentiert. Diese Beispiel-Konfigurationsdatei, zeigt das Default-Verhalten von pnp4nagios.

Listing 4.1: pnp4nagios RRD

```
RRA:AVERAGE:0.5:1:2880
RRA:AVERAGE:0.5:5:2880
RRA:AVERAGE:0.5:30:4320
RRA:AVERAGE:0.5:360:5840
RRA:MAX:0.5:1:2880
RRA:MAX:0.5:5:2880
RRA:MAX:0.5:30:4320
RRA:MAX:0.5:360:5840
RRA:MIN:0.5:1:2880
RRA:MIN:0.5:5:2880
RRA:MIN:0.5:30:4320
RRA:MIN:0.5:360:5840
```

Kommentare und Leerzeilen wurden entfernt. Wie man anhand dieser Konfigurationsdatei sehen kann, werden 4 verschiedene Auflösungen gespeichert. Als Konsolidierungsfunktionen werden Durchschnitt, Maximum und Minimum verwendet. Weiters kann man der Konfigurationsdatei */usr/local/nagios/etc/pnp/process_perfdata.cfg-*

sample entnehmen, dass für das RRD File eine Step Size von 60 Sekunden und ein Hearbeat von 8460 Sekunden für die Data Source verwendet wurde.

Für das Testsetup wird Step Size (60 Sekunden) und Hearbeat so übernommen und die Daten werden auch in mehreren Auflösungen gespeichert (3 verschiedene, 360 Minuten macht keinen Sinn, da der Test definitiv nicht so lange angelegt werden wird). Als Datenquellentyp wird *GAUGE* verwendet. Dieser Typ ist für die Zufallswerte, welche in die Datenbank geschrieben werden, am besten geeignet. Ebenso wie bei pnp4nagios, werden die Daten in RRA mit den Konsolidierungsfunktionen *AVERAGE*, *MIN* und *MAX* abgelegt. Das Shellskript welches ein RRD File anlegt, heißt *create_rrd* und erwartet den Dateinamen.

Listing 4.2: RRD Anlegen

```
1  if test -e "$1"
2  then
3      rm "$1"
4  fi
5
6  rrdtool create $1 --step 60 \
7  DS:testsource:GAUGE:8640:0:32767 \
8  RRA:AVERAGE:0.5:1:1000 \
9  RRA:AVERAGE:0.5:5:1000 \
10 RRA:AVERAGE:0.5:30:1000 \
11 RRA:MIN:0.5:1:1000 \
12 RRA:MIN:0.5:5:1000 \
13 RRA:MIN:0.5:30:1000 \
14 RRA:MAX:0.5:1:1000 \
15 RRA:MAX:0.5:5:1000 \
16 RRA:MAX:0.5:30:1000
```

Die Möglichkeiten, wie die Struktur der relationalen Datenbank aussehen könnte, sind vielfältig und es hängt natürlich vom Addon ab, welches auf der Datenbank

aufbaut. Im vorliegenden Fall wurde versucht, die zu speichernden Inhalte auf das Wesentliche zu beschränken. In einem realen Umfeld wird die Struktur wohl nicht so aussehen und es werden weit mehr Tabellen vorliegen. Da die relationalen Datenbanken, wie bereits im Abschnitt 4.3.2 auf Seite 53 besprochen, keine automatische Konsolidierung der Daten ermöglicht, um sie in unterschiedlichen Auflösungen zu speichern, müssen die Daten mit der höchsten Genauigkeit (welche in der RRD zu finden ist) gespeichert werden. Das bedeutet in der relationalen Datenbank werden Minuten-Werte gespeichert.

Die *CREATE* Statements zur Erzeugung der Tabellen in der MySQL bzw. PostgreSQL Datenbank sind in den Dateien *create_db_mysql.sql* und *create_db_psql.sql* festgehalten. Es werden 2 Tabellen angelegt, die Tabelle *perfdata*, in welcher die vom Service- oder Hostcheck ermittelten Werte gespeichert werden und exemplarisch die Tabelle *service* in welcher Informationen zum Service, in diesem Fall die Grenzwerte, festgehalten sind. In einem realen Umfeld würde es wohl zumindest auch eine eigene Tabelle für Hosts geben und die Tabellen hätten einige Attribute mehr. Dies spielt für die Tests aber keine Rolle bzw. soll nicht für ein bestimmtes Addon (wovon ja Tabellen und Attribute abhängig sind) getestet werden, sondern der Unterschied zwischen RRD und relationaler Datenbank gezeigt werden. Im folgenden die Inhalte der Dateien *create_db_mysql.sql* und *create_db_psql.sql*.

Listing 4.3: MySQL CREATE

```
1 -- create_db_mysql.sql
2
3 DROP TABLE IF EXISTS perfdata;
4 DROP TABLE IF EXISTS service;
5
6 CREATE TABLE perfdata
7 (
8     id INT AUTO_INCREMENT PRIMARY KEY,
9     sid INT,
```

```
10      val INT
11  );
12
13  CREATE TABLE service
14  (
15      id INT PRIMARY KEY,
16      min INT,
17      max INT
18  );
```

Listing 4.4: PSQL CREATE

```
1   -- create_db_psql.sql
2
3   DROP TABLE IF EXISTS perfdata;
4   DROP TABLE IF EXISTS service;
5
6   CREATE TABLE perfdata
7   (
8       id SERIAL,
9       sid INTEGER,
10      val INTEGER,
11      PRIMARY KEY (id)
12  );
13
14  CREATE TABLE service
15  (
16      id INTEGER,
17      min INTEGER,
18      max INTEGER,
19      PRIMARY KEY (id)
20  );
```

Das Attribut *sid (Service ID)* in der Tabelle *perfdata* repräsentiert den Fremdschlüssel zur Tabelle *service*.

4.4.3 Ablauf

Die Messungen werden von einem Shell-Skript (*dotest.sh*) durchgeführt, welches hier einsehbar ist und nun Schritt für Schritt erläutert wird.

Listing 4.5: Variablen

```
3  ROUNDS=10
4  STEP_SIZE=60
5  COUNTER=0
6
7  TOTAL_TIME_RRD=0
8  TOTAL_TIME_MYSQL=0
9  TOTAL_TIME_PSQL=0
10
11 RRD_DIR="rrd_files"
12
13 RRD_FILE="test"
14 MYSQL_DB="mysql_test"
15 PSQL_DB="psql_test"
16
17 MYSQL_CREATE="create_db_mysql.sql"
18 PSQL_CREATE="create_db_psql.sql"
19 RRD_CREATE="create_rrd"
20
21 NUMBER_OF_SERVICES=500;
22
23 USE_TCP=0
```

Im folgenden wird Sinn und Zweck der zu Beginn definierten Variablen erklärt.

ROUNDS Wieviele Durchläufe werden gemacht.

STEP_SIZE Sollte dem Wert der RRD Step Size entsprechen. Solange wartet das Skript nach jedem Durchlauf, damit die RRD wieder Werte annimmt.

COUNTER Der Zähler für die Durchläufe.

TOTAL_TIME_RRD Die Gesamtzeit, welche für das Einfügen in die RRD benötigt wird. Über alle Durchläufe addiert.

TOTAL_TIME_MYSQL Die Gesamtzeit, welche für das Einfügen in die MySQL DB benötigt wird. Über alle Durchläufe addiert.

TOTAL_TIME_PSQL Die Gesamtzeit, welche für das Einfügen in die PostgreSQL DB benötigt wird. Über alle Durchläufe addiert.

RRD_DIR Das Verzeichnis in dem die RRD Files abgelegt werden. Für jeden Service wird ein RRD File angelegt.

RRD_FILE Der Präfix für die RRD Files. Die RRD Files werden einfach durchnummeriert.

MYSQL_DB Der Name der MySQL Datenbank.

PSQL_DB Der Name der PSQL Datenbank.

MYSQL_CREATE Der Pfad des Files in dem die MySQL CREATEs zu finden sind.

PSQL_CREATE Der Pfad des Files in dem die PSQL CREATEs zu finden sind.

RRD_CREATE Der Pfad zu dem Skript, welches ein RRD File anlegt.

NUMBER_OF_SERVICES Die Anzahl der Services deren Werte pro Durchlauf in die Datenbanken gespeichert werden.

USE_TCP Standardmäßig auf 0. Ist diese Variable auf 1, so wird die Verbindung zur (lokalen) Datenbank per TCP aufgebaut. Dies ist deutlich langsamer und macht eigentlich keinen Sinn, wenn die Datenbank auf dem selben Server liegt.[16]

Listing 4.6: create_dbs()

```
25 create_dbs()
26 {
27     mysql $MYSQL_DB < $MYSQL_CREATE
28     psql -q $PSQL_DB < $PSQL_CREATE
29
30     if test -e $RRD_DIR
31     then
32         rm -r $RRD_DIR
33     fi
34     mkdir $RRD_DIR
35
36     I=0
37
38     while test "$I" -lt "$NUMBER_OF_SERVICES"
39     do
40         fname="$RRD_DIR/$RRD_FILE$I.rrd"
41         sh $RRD_CREATE $fname
42         let I=$I+1
43     done
44 }
```

In dieser Funktion wird *mysql* bzw. *psql* mit dem entsprechendem Datenbanknamen aufgerufen und auf *stdin* wird die Datei mit den SQL CREATEs eingegeben. Für die RRD Files wird zunächst das Verzeichnis angelegt (vorher geleert und gelöscht falls

[16] vgl. MySQL Developer Zone, 2008-12-17, 1, MySQL 5.1 Reference Manual B.1.2.2., http://dev.mysql.com/doc/refman/5.1/en/can-not-connect-to-server.html

es existiert) und dann werden darin entsprechend dem Wert der Variable *NUMBER_OF_SERVICES* RRD Files angelegt.

Listing 4.7: rrd_insert()

```
46 rrd_insert()
47 {
48     t=0
49
50     fname="$RRD_DIR/$RRD_FILE$1.rrd"
51     t=‘time -f %e rrdtool update $fname N:$RANDOM 2>&1‘
52
53     echo "$t"
54 }
```

Mittels dieser Funktion wird ein zufälliger Wert in das, dem Argument entsprechende, RRD File geschrieben. Die benötigte Zeit wird mit *time* gemessen und zurückgegeben.

Listing 4.8: mysql_insert()

```
56 mysql_insert()
57 {
58     t=0
59
60     if test $USE_TCP -eq 0
61     then
62         t=‘time -f %e mysql -e "insert into perfdata (sid,val)
               values($1,$RANDOM);" $MYSQL_DB 2>&1‘
63     elif test $USE_TCP -eq 1
64     then
65         t=‘time -f %e mysql -e "insert into perfdata (sid,val)
               values($1,$RANDOM);" -h localhost -P 3306 $MYSQL_DB
               2>&1‘
```

```
66      fi
67
68      echo "$t"
69  }
```

Mittels dieser Funktion wird ein zufälliger Wert und die Service ID (der erste Funktionsparameter) in die *perfdata* Tabelle der MySQL Datenbank geschrieben. Je nachdem ob *USE_TCP* auf 1 ist, wird die Verbindung zur Datenbank per TCP oder eben per Socket-File (die für lokale Verbindungen herkömmliche Methode) hergestellt. Wird eine Hostname und ein Port angegeben, so wird automatisch per TCP verbunden. Die benötigte Zeit wird natürlich auch hier ermittelt.

Listing 4.9: psql_insert()

```
71  psql_insert()
72  {
73      t=0
74
75      if test $USE_TCP -eq 0
76      then
77          t=‘time -f %e psql -q -c "insert into perfdata (sid,val)
                values($1,$RANDOM);" $PSQL_DB 2>&1‘
78      elif test $USE_TCP -eq 1
79      then
80          t=‘time -f %e psql -q -c "insert into perfdata (sid,val)
                values($1,$RANDOM);" -h localhost -p 5432 $PSQL_DB 2>&1‘
81      fi
82
83      echo "$t"
84  }
```

Das selbe geschieht für die PostgreSQL Datenbank. Der Aufruf von *psql* unterscheidet sich ein wenig vom Aufruf von *mysql*. Das Kommando wird mit der Option *-e*

angegeben (für execute) und der Port ist natürlich auch ein anderer.

Listing 4.10: insert_services_mysql()

```
86 insert_services_mysql()
87 {
88     t=0
89
90     if test $USE_TCP -eq 0
91     then
92         t=‘time -f %e mysql -e "insert into service (id,min,max)
               values($1,0,32767);" $MYSQL_DB 2>&1‘
93     elif test $USE_TCP -eq 1
94     then
95         t=‘time -f %e mysql -e "insert into service (id,min,max)
               values($1,0,32767);" -h localhost -P 3306 $MYSQL_DB
               2>&1‘
96     fi
97
98     echo "$t"
99
100 }
```

Die Funktion *insert_services_mysql* fügt in der *service* Tabelle einen Eintrag hinzu. Die ID wird durch den Funktionsparameter bestimmt. Es wird der *USE_TCP* beachtet und die Zeit ermittelt.

Listing 4.11: insert_services_psql()

```
102 insert_services_psql()
103 {
104     t=0
105
106     if test $USE_TCP -eq 0
```

```
107     then
108         t=‘time -f %e psql -q -c "insert into service (id,min,max)
               values($1,0,32767);" $PSQL_DB 2>&1‘
109     elif test $USE_TCP -eq 1
110     then
111         t=‘time -f %e psql -q -c "insert into service (id,min,max)
               values($1,0,32767);" -h localhost -p 5432 $PSQL_DB 2>&1‘
112     fi
113
114     echo "$t"
115 }
```

Das Einfügen in die *service* Tabelle der *PSQL* Datenbank. Analog zum Einfügen in die *service* Tabelle der *MySQL* Datenbank.

Listing 4.12: Hauptprogramm

```
117 create_dbs
118
119 # ROUNDS durchlaeufe, also ROUNDS mal kommt fuer jeden service/host
        ein performance wert daher
120 while test "$COUNTER" -lt "$ROUNDS"
121 do
122     I=0
123     ROUND_TIME_RRD=0
124     ROUND_TIME_MYSQL=0
125     ROUND_TIME_PSQL=0
126
127     # NUMBER_OF_CHEKCS host/service checks sind, die daten liefern
128     while test "$I" -lt "$NUMBER_OF_SERVICES"
129     do
```

```
130         # wenn checks das erste mal durchgefuehrt werden, fuer
                jeden check einen eintrag in service tabelle machen
131         if test "$COUNTER" -eq "0"
132         then
133             STIME=‘insert_services_mysql $I‘
134             ROUND_TIME_MYSQL=‘echo "scale=2;$ROUND_TIME_MYSQL+
                    $STIME" | bc‘
135             STIME=‘insert_services_psql $I‘
136             ROUND_TIME_PSQL=‘echo "scale=2;$ROUND_TIME_PSQL+$STIME"
                     | bc‘
137         fi
138
139         TIME_RRD=‘rrd_insert $I‘
140         ROUND_TIME_RRD=‘echo "scale=2;$ROUND_TIME_RRD+$TIME_RRD" |
                bc‘
141         TIME_MYSQL=‘mysql_insert $I‘
142         ROUND_TIME_MYSQL=‘echo "scale=2;$ROUND_TIME_MYSQL+
                $TIME_MYSQL" | bc‘
143         TIME_PSQL=‘psql_insert $I‘
144         ROUND_TIME_PSQL=‘echo "scale=2;$ROUND_TIME_PSQL+$TIME_PSQL"
                 | bc‘
145
146         let I=$I+1
147     done
148
149     TOTAL_TIME_RRD=‘echo "scale=2;$TOTAL_TIME_RRD+$ROUND_TIME_RRD"
            | bc‘
150     TIME_MYSQL=‘mysql_insert $I‘
151     TOTAL_TIME_MYSQL=‘echo "scale=2;$TOTAL_TIME_MYSQL+
            $ROUND_TIME_MYSQL" | bc‘
```

```
152     TIME_PSQL=‘psql_insert $I‘
153     TOTAL_TIME_PSQL=‘echo "scale=2;$TOTAL_TIME_PSQL+
           $ROUND_TIME_PSQL" | bc‘
154
155     let DURCHL=$COUNTER+1
156
157     echo "------------------"
158     echo "Gesamtzeit fuer Durchlauf $DURCHL:"
159     echo "RRD: $ROUND_TIME_RRD, MYSQL: $ROUND_TIME_MYSQL, PSQL:
           $ROUND_TIME_PSQL"
160     echo "------------------"
161     echo "Durschnittliche Zeit pro Check fuer Durchlauf $DURCHL:"
162     echo RRD: ‘echo "scale=3;$ROUND_TIME_RRD/$NUMBER_OF_SERVICES" |
            bc‘ MYSQL: ‘echo "scale=3;$ROUND_TIME_MYSQL/
           $NUMBER_OF_SERVICES" | bc‘ PSQL: ‘echo "scale=3;
           $ROUND_TIME_PSQL/$NUMBER_OF_SERVICES" | bc‘
163     echo "------------------"
164     echo "Gesamtzeit:"
165     echo "RRD: $TOTAL_TIME_RRD, MYSQL: $TOTAL_TIME_MYSQL, PSQL:
           $TOTAL_TIME_PSQL"
166     echo "------------------"
167
168     sleep $STEP_SIZE
169
170     let COUNTER=$COUNTER+1
171 done
172
173 echo "-----------------------------------------"
174 echo "Gesamtzeit fuer $COUNTER Durchlauefe mit jeweils
       $NUMBER_OF_SERVICES Checks:"
```

```
175 echo "RRD: $TOTAL_TIME_RRD, MYSQL: $TOTAL_TIME_MYSQL, PSQL:
      $TOTAL_TIME_PSQL"
176 echo "-----------------"
177 echo "Durchschnittliche Zeit pro Check (nach $COUNTER Durchlauefen
      mit jeweils $NUMBER_OF_SERVICES Checks):"
178 echo RRD: `echo "scale=3;$TOTAL_TIME_RRD/($COUNTER*
      $NUMBER_OF_SERVICES)" | bc` MYSQL: `echo "scale=3;
      $TOTAL_TIME_MYSQL/($COUNTER*$NUMBER_OF_SERVICES)" | bc` PSQL: `
      echo "scale=3;$TOTAL_TIME_PSQL/($COUNTER*$NUMBER_OF_SERVICES)" |
       bc`
179 echo "-----------------"
180 echo "Durchschnittliche Zeit pro Durchlauf (nach $COUNTER
      Durchlauefen mit jeweils $NUMBER_OF_SERVICES Checks):"
181 echo RRD: `echo "scale=3;$TOTAL_TIME_RRD/$COUNTER" | bc` MYSQL: `
      echo "scale=3;$TOTAL_TIME_MYSQL/$COUNTER" | bc` PSQL: `echo "
      scale=3;$TOTAL_TIME_PSQL/$COUNTER" | bc`
182 echo "-----------------------------------------"
```

Zu Beginn des Hauptprogramms wird die *create_dbs* Funktion aufgerufen. Dann läuft die äußerste Schleife so oft wie in der Variable *ROUNDS* angegeben. Dies sind also die Wiederholungen, wie oft sozusagen die einzelnen Services abgefragt werden. Also hier 10 mal im Abstand von mindestens 60 Sekunden. Die innere Schleife geht die einzelnen Services durch. Ist es der erste Durchlauf, so werden auch die Einträge in die *service* Tabelle erzeugt. Dann wird jeweils ein Eintrag in das RRD File, in die MySQL und in die PostgreSQL Datenbank gemacht. Die gemessenen Zeiten werden aufaddiert. Nach dem Durchlauf der inneren Schleife werden die aufsummierten Zeiten pro Datenbank zu der Variable (*TOTAL_TIME*) hinzugerechnet, welche die Zeiten aller Durchläufe addiert. Selbstverständlich getrennt pro Datenbank.

Nach jedem Durchlauf werden folgende Zeiten ausgegeben (jeweils pro Datenbank): Gesamtzeit für letzten Durchlauf, Durschnittszeit pro Service (auf letzten

Durchlauf bezogen) und vorläufige Gesamtzeit.

Bevor der nächste Durchlauf startet, wird 60 Sekunden (bzw. entsprechend dem Wert der *STEP_SIZE* Variable) gewartet, damit die RRD Files wieder Werte annehmen.

Sind alle Durchläufe abgeschlossen, so werden folgende Messergebnisse präsentiert (pro Datenbank): Gesamtzeit für alle Durchläufe, Durschnittszeit pro Service (Mittel über alle Durchläufe und Anzahl der Services) und die durschnittliche Zeit die ein Durchlauf benötigt hat (für das Einfügen einer Anzahl von Werten entsprechend *NUMBER_OF_SERVICES*).

4.4.4 Ergebnisse

Endergebnis

Die Endausgabe des Skripts *dotest.sh* für 500 Services und 10 Durchläufe.

Listing 4.13: Ausgabe von dotest.sh

```
101 ------------------------------------------
102 Gesamtzeit fuer 10 Durchlauefe mit jeweils 500 Checks:
103 RRD: 8.23, MYSQL: 73.49, PSQL: 469.38
104 -----------------
105 Durchschnittliche Zeit pro Check (nach 10 Durchlauefen mit jeweils
       500 Checks):
106 RRD: .001 MYSQL: .014 PSQL: .093
107 -----------------
108 Durchschnittliche Zeit pro Durchlauf (nach 10 Durchlauefen mit
       jeweils 500 Checks):
109 RRD: .823 MYSQL: 7.349 PSQL: 46.938
110 ------------------------------------------
```

Die interessantesten Werte sind wohl die durchschnittliche Dauer zum Speichern vom Wert eines Checks und die durchschnittliche Zeit für einen Durchlauf.

Das Speichern eines Werts in das RRD File geht mit 0,001 Sekunden sehr schnell und das, obwohl der Wert einige Male (mehrere RRA) gespeichert werden muss. Das Speichern in der MySQL Datenbank dauert 14 mal so lange (0,014 Sekunden) und zum Ablegen in die PostgreSQL Datenbank wird gar 93 mal so viel Zeit benötigt (0,093 Sekunden). Dieser beträchtliche Unterschied zwischen MySQL und PostgreSQL sollte einem zu denken geben. Der Grund dafür scheint jedoch schnell gefunden. MySQL verwendet als Storage-Eninge per Default *MyISAM*. Diese Storage-Eninge ist zwar sehr schnell, erfüllt jedoch einige Anforderungen an ein Datenbanksystem nicht. So kann keine Transaktionssicherheit gewährleistet werden und Foreign Key Constraints (Überprüfung von Fremdschlüsseln) sind nicht möglich. Trotzdem bleibt MyISAM eine interessante Option, denn sind diese, eigentlich fundamentalen Konzepte eines Datenbanksystems nicht von Nöten, so bleibt noch immer der Performancevorteil. Wird die Storage-Eninge auf InnoDB geändert (wodurch alle Anforderungen an ein Datenbanksystem erfüllt werden), so möchte man meinen, dass sich die Performance an die von PostgreSQL angleicht. Es ist jedoch so, dass die Speicherperfomance in die MySQL Datenbank zwar um ca. 25 Prozent abnimmt (ungefähr 0,020 Sekunden), jedoch bei weitem nicht in die Dimensionen von PostgreSQL vordringt. Will man also PostgreSQL als Datenbanksystem verwenden, so sollte man sich näher mit dem Problem und der Konfiguration von PostgreSQL beschäftigen. Und festzuhalten bleibt auch: Eine Auswahl der Storage-Engine ist bei PostgreSQL nicht möglich.[17]

Die Durschnittliche Zeit für einen ganzen Durchlauf ist, logischerweise, ca. 500 mal so groß, wie die Zeit, die für die Speicherung eines Werts benötigt wird. Aber diese Zahlen verdeutlichen schon, dass auch eine eher geringe Dauer von 0,014 Sekunden für einen Check, von Bedeutung sein kann, wenn die Daten sehr vieler Service oder Host Checks nahezu gleichzeitig eintreffen. Natürlich muss es sich schon um ein aussergewöhnlich großes System handeln, denn erstens werden die

[17] vgl. wikivs, 2008-12-17, 1, MySQL vs PostgreSQL, http://www.wikivs.com/wiki/MySQL_vs_PostgreSQL

meisten Werte nicht jede Minute abgefragt und zweitens sind 500 Werte die jede Minute eintreffen eine sehr große Zahl. Anhand der durschnittlichen Dauer für die Speicherung eines Werts eines Service oder Host Checks, kann man sich leicht für das eigene System die Speicherdauer ausrechnen.

Ergebnisse der einzelnen Durchläufe

Der Vollständigkeit halber auch die Ergebnisse der einzelnen Durchläufe.

Listing 4.14: Ausgabe von dotest.sh

```
1  -----------------
2  Gesamtzeit fuer Durchlauf 1:
3  RRD: .58, MYSQL: 13.12, PSQL: 82.13
4  -----------------
5  Durschnittliche Zeit pro Check fuer Durchlauf 1:
6  RRD: .001 MYSQL: .026 PSQL: .164
7  -----------------
8  Gesamtzeit:
9  RRD: .58, MYSQL: 13.12, PSQL: 82.13
10 -----------------
11 -----------------
12 Gesamtzeit fuer Durchlauf 2:
13 RRD: .56, MYSQL: 6.54, PSQL: 42.09
14 -----------------
15 Durschnittliche Zeit pro Check fuer Durchlauf 2:
16 RRD: .001 MYSQL: .013 PSQL: .084
17 -----------------
18 Gesamtzeit:
19 RRD: 1.14, MYSQL: 19.66, PSQL: 124.22
20 -----------------
21 -----------------
```

```
Gesamtzeit fuer Durchlauf 3:
RRD: .91, MYSQL: 6.60, PSQL: 42.02
-----------------
Durschnittliche Zeit pro Check fuer Durchlauf 3:
RRD: .001 MYSQL: .013 PSQL: .084
-----------------
Gesamtzeit:
RRD: 2.05, MYSQL: 26.26, PSQL: 166.24
-----------------
-----------------
Gesamtzeit fuer Durchlauf 4:
RRD: .42, MYSQL: 6.64, PSQL: 42.63
-----------------
Durschnittliche Zeit pro Check fuer Durchlauf 4:
RRD: 0 MYSQL: .013 PSQL: .085
-----------------
Gesamtzeit:
RRD: 2.47, MYSQL: 32.90, PSQL: 208.87
-----------------
-----------------
Gesamtzeit fuer Durchlauf 5:
RRD: 1.40, MYSQL: 6.62, PSQL: 42.59
-----------------
Durschnittliche Zeit pro Check fuer Durchlauf 5:
RRD: .002 MYSQL: .013 PSQL: .085
-----------------
Gesamtzeit:
RRD: 3.87, MYSQL: 39.52, PSQL: 251.46
-----------------
-----------------
```

```
52 Gesamtzeit fuer Durchlauf 6:
53 RRD: 1.92, MYSQL: 6.89, PSQL: 43.82
54 -----------------
55 Durschnittliche Zeit pro Check fuer Durchlauf 6:
56 RRD: .003 MYSQL: .013 PSQL: .087
57 -----------------
58 Gesamtzeit:
59 RRD: 5.79, MYSQL: 46.41, PSQL: 295.28
60 -----------------
61 -----------------
62 Gesamtzeit fuer Durchlauf 7:
63 RRD: .54, MYSQL: 6.83, PSQL: 43.55
64 -----------------
65 Durschnittliche Zeit pro Check fuer Durchlauf 7:
66 RRD: .001 MYSQL: .013 PSQL: .087
67 -----------------
68 Gesamtzeit:
69 RRD: 6.33, MYSQL: 53.24, PSQL: 338.83
70 -----------------
71 -----------------
72 Gesamtzeit fuer Durchlauf 8:
73 RRD: .64, MYSQL: 6.69, PSQL: 43.46
74 -----------------
75 Durschnittliche Zeit pro Check fuer Durchlauf 8:
76 RRD: .001 MYSQL: .013 PSQL: .086
77 -----------------
78 Gesamtzeit:
79 RRD: 6.97, MYSQL: 59.93, PSQL: 382.29
80 -----------------
81 -----------------
```

```
82 Gesamtzeit fuer Durchlauf 9:
83 RRD: .60, MYSQL: 6.92, PSQL: 43.65
84 -----------------
85 Durschnittliche Zeit pro Check fuer Durchlauf 9:
86 RRD: .001 MYSQL: .013 PSQL: .087
87 -----------------
88 Gesamtzeit:
89 RRD: 7.57, MYSQL: 66.85, PSQL: 425.94
90 -----------------
91 -----------------
92 Gesamtzeit fuer Durchlauf 10:
93 RRD: .66, MYSQL: 6.64, PSQL: 43.44
94 -----------------
95 Durschnittliche Zeit pro Check fuer Durchlauf 10:
96 RRD: .001 MYSQL: .013 PSQL: .086
97 -----------------
98 Gesamtzeit:
99 RRD: 8.23, MYSQL: 73.49, PSQL: 469.38
100 -----------------
```

Die Messungen aus dem ersten Durchgang sind mit Vorsicht zu genießen, da hier für die relationalen Datenbanken noch die Einträge in die *service* Tabelle gemacht werden müssen, wenn der Service erstmals Werte liefert.

Verteilung der Schwankungen von RRD

Wie im vorigen Abschnitt zu erkennen ist, ist die Zeit, welche pro Durchlauf benötigt wird, bei RRDtool starken Schwankungen ausgesetzt. Um zu ermitteln, ob die Schwankungen gewissen Regelmäßigkeiten gehorchen, wurde der beschriebene Test (mit zehn Durchläufen) wiederholt durchgeführt (insgesamt acht mal). Die Ergebnisse sind dem Diagramm auf Seite 79 zu entnehmen. Wie man in diesem Diagramm

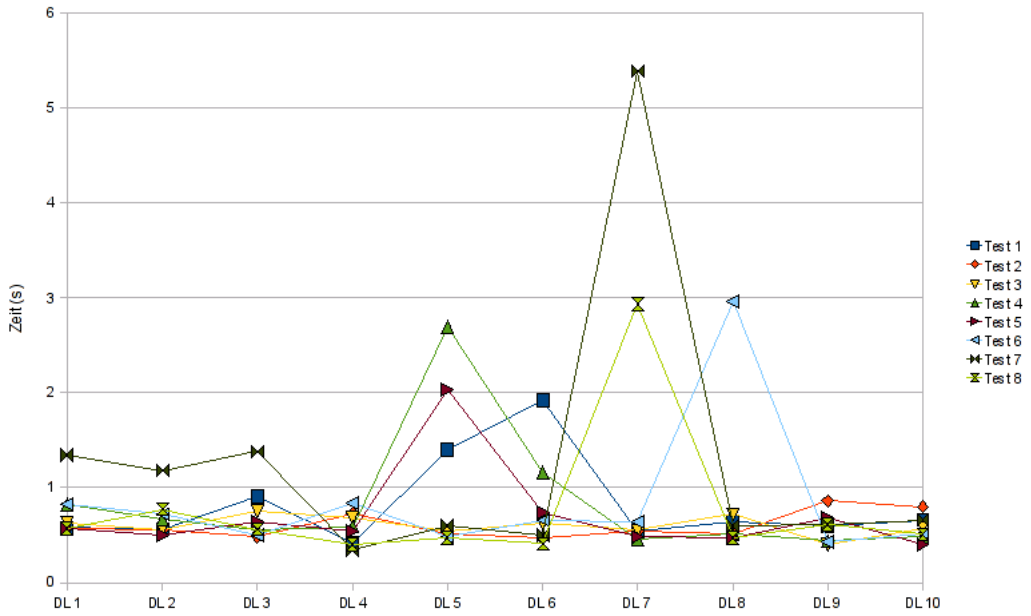


Abbildung 4.1: RRDtool Performance Tests

erkennen kann, sind die *Ausreißer* nach oben hin scheinbar zufällig verteilt und bei manchen Tests auch gar nicht vorhanden. Da diese Ausreißer jedoch frühstens beim fünften Durchlauf auftreten, kann man in Betracht ziehen, dass diese durch die Konsolidierung innerhalb der RRD und den Eintrag in weitere RRA zustande kommen. Denn wie im Abschnitt 4.4.2 auf Seite 60 erläutert, wird laut Defintion der RRD erstmals bei 5 PDP eine Konsolidierung durchgeführt. Jedoch müsste auch nach zehn Durchläufen wieder eine Konsolidierung und ein Eintrag in das RRA, welches CDP generiert aus 5 PDP aufnimmt, erfolgen. Hier sind jedoch kein weiteres Mal Ausreißer zu beobachten. Damit die Konsolidierung wiederholt stattfindet und so das Verhalten genauer untersucht werden kann, sind Tests mit mehr Durchläufen notwendig. Diese intensiveren Tests sind im Abschnitt 4.4.5 auf Seite 79 beschrieben.

4.4.5 Tests mit adaptierten Parametern (1)

Um weitere Aussagen machen zu können, ist ein Testlauf über einen längeren Zeitraum notwendig. Interessant sind dabei folgende Dinge:

- Verändert sich die Dauer pro Durchlauf wenn der Test länger läuft und wenn ja, wo macht es sich stärker bemerkbar. Bei RRDtool oder bei der relationalen Datenbank?
- Macht sich die Konsolidierung, die in regelmäßigen Abständen stattfindet, zeitlich bemerkbar. Also dauern Durchgänge, in dessen Umgebung eine Konsoldierung stattfindet, deutlich länger?

Anpassungen

Es wird nun mit *50 Service- bzw. Hostchecks* getestet und *auf PostgreSQL verzichtet*, da MySQL ohnehin als die für diese Anwendung geeignetere Lösung erscheint. Diese Maßnahmen werden getroffen, um den Ablauf zu beschleunigen. Weiters wird nun angenommen, dass die Resultate der *Checks in zufälliger Reihenfolge* ankommen. Die Reihenfolge, in der in die verschiedenen RRD Files geschrieben wird, ist nun bei jedem Durchlauf anders. Da der Test nun mit *1000 Durchläufen* durchgeführt wird, enstehen größere Datenmengen und es ist nun auch von großer Bedeutung, dass keine Bewegegung des Schreib-/Lesekopfes der Platte verursacht wird, weil abwechselnd in RRD Files und die relationale Datenbank geschrieben wird. Daher werden *2 seperate Tests* durchgeführt. Einmal für RRDtool und einmal für MySQL. Die *Wartezeit nach jedem Durchlauf wurde entfernt* und durch ein *sleep* nach jedem Einfügen in die RRD bzw. RDB ersetzt. Diese Wartezeit beträgt hier eine Sekunde. Würden die Daten direkt hintereinander geschrieben werden (was kaum der Realität entspricht), würde das zwar den Testablauf beschleunigen, aber auch die einzelnen Einfüge-Operationen verfälschen. Im folgenden ist das adaptierte Shell-Skript jeweils für RRDtool bzw. MySQL zu sehen.

Listing 4.15: Hauptprogramm (Angepasst für reinen RRDtool Test)

```
117 create_dbs
118
119 while test "$COUNTER" -lt "$ROUNDS"
```

```
120 do
121     I=0
122     ROUND_TIME_RRD=0
123
124     for I in `./randval`
125     do
126         TIME_RRD=`rrd_insert $I`
127         ROUND_TIME_RRD=`echo "scale=2;$ROUND_TIME_RRD+$TIME_RRD" |
                bc`
128         sleep 1
129     done
130
131     TOTAL_TIME_RRD=`echo "scale=2;$TOTAL_TIME_RRD+$ROUND_TIME_RRD"
            | bc`
132
133     echo -n "$ROUND_TIME_RRD,"
134
135     let COUNTER=$COUNTER+1
136 done
```

Listing 4.16: Hauptprogramm (Angepasst für reinen MySQL Test)

```
101 create_dbs
102
103 while test "$COUNTER" -lt "$ROUNDS"
104 do
105     I=0
106     ROUND_TIME_MYSQL=0
107
108     for I in `./randval`
109     do
110         if test "$COUNTER" -eq "0"
```

```
111         then
112             STIME=‘insert_services_mysql $I‘
113             ROUND_TIME_MYSQL=‘echo "scale=2;$ROUND_TIME_MYSQL+
                    $STIME" | bc‘
114         fi
115
116         TIME_MYSQL=‘mysql_insert $I‘
117         ROUND_TIME_MYSQL=‘echo "scale=2;$ROUND_TIME_MYSQL+
                $TIME_MYSQL" | bc‘
118
119         sleep 1
120     done
121
122     TOTAL_TIME_MYSQL=‘echo "scale=2;$TOTAL_TIME_MYSQL+
            $ROUND_TIME_MYSQL" | bc‘
123
124     echo -n "$ROUND_TIME_MYSQL,"
125
126
127     let COUNTER=$COUNTER+1
128 done
```

Zudem ist es nun wichtig, die Größe der RRD Files (welche ja zu Beginn komplett geschrieben werden) realitätsnahe zu wählen. Die Struktur sieht nun folgendermaßen aus (stark an die von *pnp4nagios* angelehnt):

Listing 4.17: RRD Struktur

```
6 rrdtool create $1 --step 60 \
7 DS:testsource:GAUGE:8640:0:32767 \
8 RRA:AVERAGE:0.5:1:2880 \
9 RRA:AVERAGE:0.5:5:2880 \
```

```
10 RRA:AVERAGE:0.5:30:4320 \
11 RRA:MIN:0.5:1:2880 \
12 RRA:MIN:0.5:5:2880 \
13 RRA:MIN:0.5:30:4320 \
14 RRA:MAX:0.5:1:2880 \
15 RRA:MAX:0.5:5:2880 \
16 RRA:MAX:0.5:30:4320
```

Ergebnisse

Wichtig ist festzuhalten, dass die Ergebnisse dieser Tests wegen der stark abgeänderten Parameter nicht mit denen der ursprünglichen Tests zu vergleichen sind. Weiters ist es so, dass auch bei diesen Tests keine großen Datenmengen entstehen bzw. entstanden sind. Ein Durchlauf dauert ungefähr 50 Sekunden (50 Checks, nach jedem Check eine Sekunde Wartezeit). Dies bedeutet, dass der Test mit 1000 Durchläufen ca. 14 Stunden beansprucht. Trotzdem entstehen nur RRD Files mit einer Gesamtgröße von ca. 12 MB. Möchte man deutlich größere Datenmengen erzeugen (im Gigabyte Bereich), so muss der Test schon sehr lange dauern. Es ist im Rahmen dieser Arbeit nicht möglich solch umfangreiche Tests durchzuführen. Wie trotzdem große Datenmengen in einem annähernd realistischem Umfeld erzeugt werden können, wird im nächsten Abschnitt (Abschnitt 4.4.6 auf Seite 85) beschrieben. Die Ergebnisse der beiden Testläufe sind im Diagramm auf Seite 84 ersichtlich. Entlang der X-Achse sind die einzelnen Durchläufe eingetragen. Die Y-Achse repräsentiert die Zeitdauer eines Durchlaufs. Zu erkennen ist, dass sich weder bei RRDtool (blau), noch bei MySQL (rot), die beanspruchte Zeit pro Durchlauf über die Testdauer verändert. Dies ist bei RRDtool relativ naheliegend, da eben die RRD Files von Anfang an existieren und sie ihre Größe im Laufe des Tests nicht verändern. Es ist bei RRDtool sogar so, dass die durchschnittliche Zeit pro Durchlauf für die erste Hälfte des Tests (also für die ersten 500 Durchläufe) bei 0,53 Sekunden liegt und für die zweite Hälfte bei 0,523.

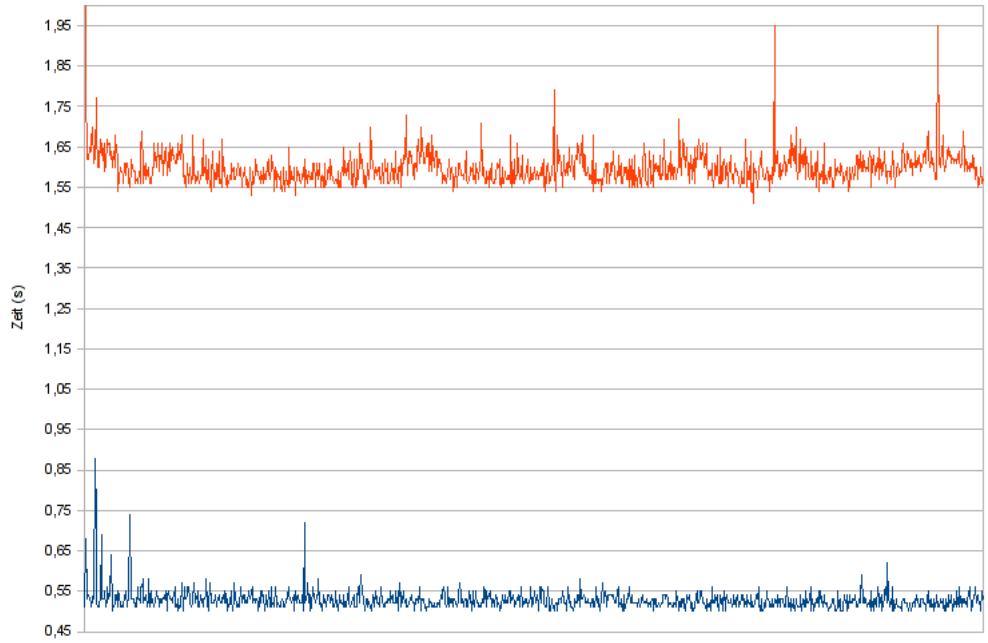


Abbildung 4.2: Tests mit adaptieren Parametern (1)

Die Performance nimmt also definitiv nicht ab. Bei MySQL wird es so sein, dass die Datenmengen pro Tabelle einfach zu gering sind, um eine signifikante Performanceverschlechterung zu bemerken. Die durschnittliche Zeit pro Durchlauf ist in der ersten Hälfte gleich wie in der zweiten Hälfte. Weiters ist es so, dass bei MyISAM die Records einer Tabelle in Einfüge-Reihenfolge gespeichert werden. Daher sollte der zeitliche Aufwand, welcher beim Einfügen entsteht, bei größeren Tabellen ohnehin keiner starken Veränderung unterworfen sein. Die Konsolidierung innerhalb der RRD, welche bei 5 und 30 PDP stattfindet, ist zeitlich nicht auffällig. Tatsächlich ist es so, dass die Einfügedauer bei MySQL stärkeren, wenn auch kaum erwähnenswerten, Schwankungen unterworfen ist. Die durschnittliche Dauer pro Durchlauf über die gesamte Testdauer liegt bei RRDtool bei 0,526 Sekunden. RRDtool speichert die Performance-Daten von 50 eingehenden Checks also innerhalb von 0,526 Sekunden. Mit MySQL dauert ein Durchlauf 1,6 Sekunden. Beides liegt also in einem akzeptablen Bereich. Man sollte jedoch bedenken, dass 50 Checks eine relativ geringe Menge sind. 500 und mehr eingehende Servicechecks sind keine Seltenheit. Was zumindest

eine Verzehnfachung der Speicherdauer bedeuten würde.

4.4.6 Tests mit adaptierten Parametern (2)

Um RRDtool mit größeren Datenmengen (größere RRD Files) zu testen, stehen zwei Möglichkeiten zur Verfügung. Einerseits kann beim Anlegen der RRD Files die Größe der RRA (also die Anzahl der CDP die gehalten werden) überdimensioniert werden. Es ist zwar im Rahmen der Tests nicht möglich die RRA vollständig zu beschreiben, aber da die RRD Files von Anfang an mit voller Größe angelegt werden (mit UNKNOWN Werten gefüllt) wird durchaus das Ziel von größeren RRD Files erreicht. Die andere Möglichkeit ist, die Step Size der RRD so zu verringern, dass die Frequenz mit der geschrieben wird erhöht werden kann. Die zweite Möglichkeit ist jedoch absolut fern der Realität, denn die Step Size wird kaum unter 60 Sekunden liegen. Die internen Abläufe von RRDtool würden sich durch eine Verringerung der Step Size deutlich anders verhalten, so, dass die Tests keine Aussagekraft hätten.

Anpassungen

Bei den nachfolgenden Tests werden also größere RRD Files angelegt. Die Struktur der RRD Files sieht folgendermaßen aus.

Listing 4.18: RRD Struktur

```
6  # Minutenwerte ueber 6 Monate speichern (60 * 24 * 30 * 6)
7  # 5-Minutenwerte ueber 5 Jahr speichern (60 * 24 * 365 * 5 / 5)
8  # 30-Minutenwerte ueber 10 Jahre speichern (60 * 24 * 365 * 10 /
       30)
9
10 rrdtool create $1 --step 60 \
11 DS:testsource:GAUGE:8640:0:32767 \
12 RRA:AVERAGE:0.5:1:259200 \
13 RRA:AVERAGE:0.5:5:525600 \
```

```
14 RRA:AVERAGE:0.5:30:175200 \
15 RRA:MIN:0.5:1:259200 \
16 RRA:MIN:0.5:5:525600 \
17 RRA:MIN:0.5:30:175200 \
18 RRA:MAX:0.5:1:259200 \
19 RRA:MAX:0.5:5:525600 \
20 RRA:MAX:0.5:30:175200
```

Obwohl die Daten über sehr lange Zeiträume gespeichert werden (bis zu 10 Jahren!) bleibt die Größe eines RRD Files eher gering. Die RRD Files weiter zu vergrößern würde keinen Sinn machen, da man selbst mit diesen Einstellungen wohl schon über die Grenzen der in Realität gewünschten Zeiträume geht. Ein RRD File ist somit 22 MB groß. In Summe werden 1,1 GB Speicherplatz beansprucht. Die weiteren Parameter und das Hauptprogramm sind ident zu den Tests in Abschnitt 4.4.5 auf Seite 79. Problematisch ist nun, dass die Möglichkeiten, welche RRDtool bietet um größere Datenmengen zu simulieren, bei MySQL nicht zur Verfügung stehen. Bei der relationalen Datenbank wird der Speicherplatz natürlich nicht von Anfang an belegt, sondern erst nach und nach beschrieben. Daher können hier keine weiteren Tests durchgeführt werden und es ist leider auch kein Vergleich möglich. Von Interesse ist bei diesem Test nur, wie sich die Performance von RRDtool im Vergleich zu den vorigen Tests mit den kleineren RRD Files verhält. Wie eine vergleichbare Entwicklung bei MySQL wäre, ist hier nicht ermittelbar.

Ergebnisse

Die Ergebnisse sind schnell erläutert, denn sie ähneln denen aus dem vorigen Abschnitt. Die durschnittliche Zeit pro Durchlauf liegt weiterhin bei ungefähr 0,525 Sekunden. Eine Vergrößerung der RRD-Files (von ca. 220 KB auf ca. 22 MB) bedeutet also keine Performance-Einbuße. Wie bereits erläutert, ist es absolut irrelevant ob sich die Performance verschlechtern würde, wenn man die RRD-Files weiter ver-

größert. Die gewählte Anzahl der CDP in den RRA ist absolut an bzw. über der Grenze eines realen Umfelds.

Kapitel 5

Fazit

5.1 Vergleich RRD und relationale DB

Grundsätzlich muss man sagen, dass die Round-Robin-Datenbank, RRDtool im speziellen, die deutlich bequemere Lösung ist, sucht man eine Speichermöglichkeit für die Performance-Daten. RRDtool ist genau auf die vorliegende Problemstellung zugeschnitten und es bedeutet kaum Aufwand, möchte man diese Lösung einsetzen. Zudem ist die Performance sensationell gut und unterliegt auch bei großen Datenmengen keinen erkennbaren Einbußen. Die gute Performance ist auf die Struktur der RRD zurückzuführen, denn beim Schreiben wird einfach sequentiell vorgegangen. Aber auch wahlfreie Zugriffe stellen kein Problem dar, da die Position einfach berechnet werden kann. Die Darstellung von Daten in Diagrammen, was zumeist der Hintergrund sein wird, warum man die Performance-Daten überhaupt aufzeichnet, ist mit RRDtool einfach, schnell und doch flexibel möglich.

Klar ist, dass eine relationale Datenbank (RDB) in den erwähnten Punkten mit RRDtool nicht mithalten kann. Denn die RDB ist ein allgemein gehaltenes Konzept und zielt nicht nur auf Speicherung und Abrufen einer bestimmten Art von Daten ab. Aber genau hier liegt auch die Stärke der RDB. Die Flexibilität. Diese Flexibilität betrifft zum einem die Datenstruktur - man ist nicht darauf beschränkt Zahlenwerte

zu speichern, leicht kann man zusätzliche Informationen damit verbinden. Weiters ist man beim Zugriff auf die Daten weit flexibler. Denn alle geläufigen Programmiersprachen ermöglichen einen mehr oder weniger komfortablen (Netzwerk-)Zugriff auf die Daten. Flexibel ist man auch, was spätere Änderungen und Ergänzungen hinsichtlich der Stuktur und die Integration der Daten in bestehende Umgebungen betrifft.

Die Möglichkeiten welche eine RDB bietet sind deutlich umfangreicher, doch bedeutet es auch erst einmal einiges an Entwicklungsaufwand, um Ähnliches zu erreichen, was RRDtool schon von sich aus anbietet. Dies betrifft zum einen das Erstellen von Diagrammen und zum anderen, und dies ist die große Stärke von RRDtool, die Konsolidierung der Daten. Um die Datenmengen gering zu halten, wird man in großen Umgebungen auch hier die Daten, ähnlich RRDtool, zusammenfassen wollen. Man will also unterschiedliche Auflösungen der Daten für verschiedene Zeiträume erreichen. Zwar bedeutet es einen nicht zu unterschätzenden Aufwand, jedoch kann man dann auch das zwangsläufige Überschreiben der Daten umgehen. Die in diesem Anwendungsbereich des öfteren kritisierte Grundeigenschaft der RRD.

Was man bei all den Möglichkeiten, welche eine relationale Datenbank bietet, nicht vergessen sollte, ist die Performance. Zwar liefert auch die RDB eine akzeptable Performance (siehe Tests im Abschnitt 4.4.5 auf Seite 79), jedoch muss man einerseits beachten, dass mit einer geringen Anzahl an Checks getestet wurde und andererseites muss man sich bewusst sein, dass wenn die RDB um die Features von RRDtool ergänzt wird (vor allem die Konsolidierung der Daten), die Performance weiter leiden wird. Es wird natürlich vom Umfeld abhängen, wie viel Performance man überhaupt braucht, von *Live* Diagrammen wird man dann jedoch weit entfernt sein.

Zusammengefasst kann man sagen, dass die relationale Datenbank jene Lösung ist, welche einem mehr und flexiblere Möglichkeiten bietet, es jedoch einen gewis-

sen Aufwand bedeutet, die Stärken von RRDtool nachzuahmen. Die Performance, welche immer deutlich schlechter als bei der Round-Robin-Datenbank sein wird, sollte man jedoch, vor allem bei der funktionalen Ergänzung der Softwarelösung, immer im Auge behalten und abschätzen wieviel Performance notwendig ist.

5.2 Weitere Betrachtungen

Weitere Betrachtungen, welche diese Arbeit nicht abdeckt, betreffen vor allem die Performance der relationalen Datenbank. Einerseits stellt sich die Frage, wie entwickelt sich die Dauer, welche notwendig ist um Daten in die Datenbank abzulegen, wenn auch hier eine Konsolidierung der Daten stattfindet. Dies ist natürlich stark von der Softwarelösung abhängig und hier kann sicherlich viel falsch aber auch richtig gemacht werden. Konkret war es bei dieser Arbeit nicht möglich, die Performance in einem solchen Fall zu testen. Die einzige bekannte Softwarelösung, welche so vorgeht, ist der *NETWAYS Grapher v2* und dieser befindet sich noch in Entwicklung. Ohnehin wäre es wenig sinnvoll das Verhalten von nur diesem einen Produkt zu analysieren, da man keinen Vergleich hat und so die Ergebnisse der Performance-Tests kaum Aussagekraft hätten.

Eine weitere Betrachtung würde sich die Entwicklung der Speicher-Performance der RDB über lange Zeiträume verdienen. Wie im Abschnitt 4.4.6 auf Seite 85 erläutert, zeigt RRDtool hier keine Schwächen. Es war jedoch, wie ebenfalls im Abschnitt 4.4.6 beschrieben, nicht sinnvoll möglich die Ergebnisse dieser Entwicklung für eine relationale Datenbank zu ermitteln. Um hier zuverlässige Resultate zu erhalten, müssten Tests stattfinden, welche über Wochen und Monate laufen. Also in einem nahezu realen Umfeld.

Was dem aufmerksamen Leser dieser Arbeit nicht entgangen sein sollte ist, dass die Analyse der Lese-Performance stark vernachlässigt wurde. Dies hat zwei Gründe. Erstens geht man davon aus, dass das Lesen von Festplatte deutlich schneller von statten geht, als das Schreiben. Der Schreibvorgang macht also den Haupt-

anteil der verstrichenen Zeit im Gesamtprozess aus. Zweiter Grund ist, dass der interessantere und bedeutendere Teil beim Auslesen der Daten, wohl nicht das Lesen aus der Datenbank ist, sondern die entsprechende Aufbereitung danach. Dies ist genauso, wie auch der Zugriff auf die Daten der Datenbank, also wie werden welche Daten abgerufen, stark von der jeweiligen Implementierung abhängig und streift das eigentliche Thema dieser Arbeit, den Vergleich von RDB und RRD, nur. Eine vernünftige Abhandlung dieser Problematik würde sich eine eigene Arbeit verdienen.

5.3 Ausblick

Die Firma *NETWAYS* hat mit dem *NagiosGrapher* (siehe Abschnitt 3.2.1 auf Seite 23) ein anerkanntes und beliebtes AddOn für das Graphing und Trending von Performance Daten. Dieses AddOn speichert, wie bereits erwähnt, die Daten in einer RRD. Trotz dieses bestehenden Produkts, hat sich NETWAYS entschlossen ein neues Graphing AddOn zu entwickeln. Den *NETWAYS Grapher v2* (siehe Abschnitt 3.2.1 auf Seite 23). Der NETWAYS Grapher v2 basiert, im Gegensatz zum alten NagiosGrapher, auf einer relationalen Datenbank. Alleine diese Entwicklung ist schon bemerkenswert, denn NETWAYS ist ein Unternehmen, dass nicht nur durch den NagiosGrapher, sondern auch durch weitere Nagios AddOns und andere Produkte bzw. Dienstleistungen, in diesem Bereich viel Erfahrung sammeln konnte und daher ist diese Entwicklung mit Sicherheit ein wohl überlegter Schritt. Wie im Abschnitt 3.2.1 beschrieben, bietet der NETWAYS Grapher v2 auch andere Weiterentwicklungen, der Umstieg bzw. der Trend zur relationalen DB ist aber wohl die einschneidenste Änderung.

Der NETWAYS Grapher v2 (NG2) ist noch in Entwicklung und liegt aktuell nur in einer sehr frühen Version vor (0.0.1-prealpha). Der NG2 nimmt Performance-Daten über 3 verschiedene Schnittstellen an. Über eine Datei, über einen Network Socket oder über das Pipe Interface (In der Nagios Konfigurationsdatei kann für die

Performance-Daten Verarbeitung über Datei der Pipe-Mode aktiviert werden). Die übernommenen Daten werden dann verarbeitet und in der relationalen Datenbank gespeichert (aktuell nur MySQL).

Es wird spannend sein zu beobachten, wie das fertige Produkt schlussendlich aussieht und wie sich der NETWAYS Grapher v2 im Vergleich zu RRD basierten Graphern, wie dem NagiosGrapher oder PNP4Nagios, schlägt. Die funktionalen Vorteile eines Graphing AddOns, basierend auf einer relationalen Datenbank, wurden in dieser Arbeit ausreichend erläutert und daher wird viel von der Frage abhängen, wie gut die Performance des NG2 bei der Verarbeitung eines hohen Datenaufkommens ist.

Abbildungsverzeichnis

Literaturverzeichnis

[1] Nico Bagari. Brainypdm, 2008. http://www.brainypdm.org.

[2] Soren Dossing. nagiosgraph, 2008. http://nagiosgraph.wiki.sourceforge.net.

[3] en.wikipedia. Wikipedia, 2009. http://en.wikipedia.org.

[4] Ethan Galstad. *Nagios Version 3.x Dokumentation*, 2007.

[5] Chris MacKinnon. Processor.com, 2009. http://www.processor.com/.

[6] nagios wiki. Nagios-wiki, 2008. http://www.nagios-wiki.de.

[7] nagiosexchange. Nagiosexchange, 2008. http://www.nagiosexchange.org.

[8] netways. Netways gmbh, 2008. http://www.netways.de.

[9] Tobias Oetiker. Rrdtool, 2008. http://oss.oetiker.ch/rrdtool/.

[10] perfparse. Perfparse wiki area, 2008. http://perfparse.de/tiki-index.php.

[11] pnp4nagios. pnp4nagios.org, 2008. http://www.pnp4nagios.org/.

[12] wikipedia. Wikipedia, 2008. http://de.wikipedia.org.

[13] wikivs. Wikivs, 2008. http://www.wikivs.com/wiki/Main_Page.

[14] MySQL Developer Zone. Mysql developer zone, 2008. http://dev.mysql.com/.