TECHNISCHE UNIVERSITÄT MÜNCHEN

FAKULTÄT FÜR INFORMATIK

Bachelorarbeit in Informatik

Design und Entwicklung einer skalierbaren Monitoringumgebung für Clustersysteme

Tanja Kühn

TECHNISCHE UNIVERSITÄT MÜNCHEN

FAKULTÄT FÜR INFORMATIK

Bachelorarbeit in Informatik

Design und Entwicklung einer skalierbaren Monitoringumgebung für Clustersysteme

Design and development of a scalable monitoring environment for cluster systems

Autor:	Tanja Kühn
Aufgabensteller:	Prof. Dr. Martin Schulz
Betreuer:	M. Sc. Dai Yang
Abgabedatum:	15.Juli 2018

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

München, Tanja Kühn

Zusammenfassung

Um für eine Netzwerkumgebung ein geeignetes Monitoring-System zu finden, gibt es verschiedenste Kriterien, welche einem helfen, eine lokal passende Lösung zu ermitteln. In dieser Arbeit werden insgesamt acht Monitoringsysteme betrachtet und anhand der Kriterien Erweiterbarkeit, Skalierbarkeit, Kostenpunkt, sowie des Zugriffes auf die Monitoringdaten untersucht. Aufgrund der daraus resultierenden Ergebnisse kommen zwei dieser Systeme, Zabbix und DataCenter-DataBase, für den geplanten Einsatzzweck in Frage und werden weiter untersucht. Dazu wird auf die Installation, aufgetretene Probleme währenddessen, und Messungen zur Performance des Hostsystems eingegangen. Abschließend zeigt sich keine klare Tendenz, welches dieser beiden Monitoringsysteme besser geeignet ist, sodass lediglich eine Empfehlung gegeben werden kann, die davon abhängt, welche positiven Eigenschaften für den jeweiligen Einsatzzweck passender sind.

Inhaltsverzeichnis

1 Einführung

Am Lehrstuhl für Rechnerarchitektur und parallele Systeme wird aktuell ein neues Forschungsnetz aufgebaut. Dazu ist es unter Anderem sehr wichtig ein System zu haben, welches automatisch die einzelnen Komponenten überwacht und im Ernstfall die zuständigen Mitarbeiter alarmiert. Diese Arbeit beschäftigt sich damit, für die genannte Situation ein geeignetes System zu finden, welches das Netzwerk inklusive der geplanten Cluster überwachen kann. Dazu werden im Folgenden für diese Arbeit wichtige Begriffe näher erläutert. Anschließend werden die Kriterien und Einschränkungen für dieses Projekt geklärt, wonach in den darauffolgenden Kapiteln zuerst einige mögliche Monitoringsysteme beschrieben und näher eingegrenzt werden. Die verbleibenden möglichen Systeme werden installiert und Messungen unterzogen, die zur Entscheidung, welches davon zum Einsatz kommen soll, mit einbezogen werden. Abschließend werden diese Messungen für eine Empfehlung herangezogen, welches System für dieses neue Forschungsnetz verwendet werden sollte und in welcher Konfiguration.

1.1 Terminologie

Monitoring Auf der Webseite des Online-Wörterbuches Duden, welche alle Wörter der deutschen Sprache abbildet und erklärt, wird der Begriff **Monitoring** als eine „[Dauer]beobachtung [eines bestimmten Systems]“beschrieben. Direkte Synonyme dazu sind *Beobachtung*, sowie *Kontrolle*, welche sich auf die Herkunft des Wortes zurückführen lassen.
Monitoring kommt ursprünglich aus dem englischen Sprachraum und ist abgeleitet von dem Verb „to monitor“, also „beobachten“oder „kontrollieren“. Mittlerweile wurde der Begriff *Monitoring* in die deutsche Sprache integriert. Bezogen auf das Thema dieser Arbeit, geht es also darum, ein bestimmtes System zu überwachen.[3]

Cluster Dazu findet man im Online-Wörterbuch Duden gleich mehrere Bedeutungen. Unter Anderem sind *Cluster* in Bezug auf die Musik Klanggebilde, welche durch Übereinanderstellen kleiner Intervalle entstehen. In der Sprachwissenschaft wird es als eine ungeordnete Menge semantischer Merkmale eines Begriffes bezeichnet. Eine weitere Bedeutung, die der Duden führt, ist in Bezug zur „Fachsprache“genannt und

benennt Cluster als eine Menge von Einzelteilchen, welche als einheitliches Ganzes zu betrachten sind.
Der Wortursprung liegt auch hier im Englischen und hat dort die Bedeutung „Büschel“oder „Menge“. Im weiteren Verlauf der Arbeit ist mit dem Begriff *Cluster* ein Zusammenschluss einzelner Rechenknoten gemeint, welche eine günstige Alternative zu Mehrprozessorsystemen darstellen. [4][16]

Metrik Damit ist ein bestimmtes Kriterium benannt, nach dem ein System überwacht werden soll. Das kann zum Beispiel die Auslastung des Festplattenspeichers sein, sodass man über das Monitoring gewarnt wird, wenn dieser über einen bestimmten Prozentsatz mit Daten belegt ist.

1.2 Hintergrund

Das Thema für diese Arbeit ist mit dem Aufbau des neuen Forschungsnetzes für den Lehrstuhl für Rechnerarchitektur und parallele Systeme entstanden. So gibt es auch von dieser Seite einige Punkte, die bei der Wahl eines geeigneten Monitoringsystems beachtet werden müssen. Die wichtigste Anforderung daraus ist eine *Erweiterbarkeit* des Systems. Das heißt, es muss die Möglichkeit bestehen, eigene Kriterien für das Monitoring festlegen zu können, welche dann ebenfalls mit überwacht werden. Es ist also darauf zu achten, dass das System direkt eine solche Möglichkeit der Erweiterung zulässt, oder eine Lizenz vorliegt, welche das Erweitern des Programmcodes zulässt.
Ebenfalls ist es nötig, ein System zu finden, bei dem die *Daten* aus der Monitoringdatenbank *exportiert* werden können, sodass man die Datensätze auch mit Werten vergleichen kann, die nicht vom Monitoring stammen.
Um für zukünftiges Wachstum der Netzwerkinfrastruktur gerüstet zu sein, sollte auch *Skalierbarkeit* gegeben sein. Grund hierfür ist, dass das Forschungsnetz gerade erst aufgebaut und kontinuierlich erweitert wird. Besonders liegt hierbei das Augenmerk auf möglichen neuen Clustern. Das ausgewählte Monitoring-System sollte daher möglichst geringe Systemeinbußen nach sich ziehen, ob nun 50 oder über 1000 Einheiten überwacht werden.
Um die Auswahl an möglichen Programmen vorab einzugrenzen, werden in dieser Arbeit nur kostenfreie Lösungen betrachtet.

1.3 Verwandte Themen

Es gibt einige Artikel, in denen untersucht wird, wie ein bestimmtes Monitoringsystem sich verhält, oder welchen Einfluss es auf das Betriebssystem und dessen Performanz hat. Einer davon ist *„The ganglia distributed monitoring system: design, implementatin, and experience“*. Darin wird erörtert, wie sich Ganglia anhand der Punkte Skalierbarkeit, Robustheit, Erweiterbarkeit, Verwaltbarkeit, Portabilität und Overhead auf verschiedenen High Performance Clustern verhält. Untersucht werden dazu je zwei Clusterverbände, sowie Einzelcluster auf Nutzung der CPU, des Speichers, sowie des I/O Overhead.[13]
In einer weiteren Arbeit, *„PerfSONAR: A Service Oriented Architecture for Multi-domain Network Monitoring“*, wird ein System dargestellt, welches den Fokus auf das Monitoring von Netzwerken mit mehreren Domains legt.[11]
In dieser Arbeit soll jedoch ein Monitoringsystem für eine Umgebung gefunden werden, in der die Anzahl der zu überwachenden Hosts mit der Zeit steigen wird.

2 Wahl der Monitoringumgebung

2.1 Mögliche Systeme

Welche Systeme können hier nun zum Einsatz kommen? Nach anfänglicher Recherche fanden sich für die gestellten Anforderungen unter anderem folgende Möglichkeiten:

PARMON ist ein für Cluster-Systeme entwickeltes Monitoring-Programm und bietet die Möglichkeit auf verschiedenen Ebenen zu überwachen. Das ist zum Beispiel ein einzelner Knoten, eine Gruppe von Knoten oder auch das komplette Cluster. Dabei werden die Resssourcenauslastung und Aktivitäten einzelner Komponenten kontrolliert und als ein komplettes Systemabbild dargestellt. *PARMON* bietet zudem ein Webinterface, um das Netzwerk auch aus der Ferne überwachen zu können.
Jedoch existiert keine programmseitige Möglichkeit, eigene Metriken zu definieren. [5]

Nagios Hier wird als kostenlose Variante das Nagios Core angeboten. Diese Version ermöglicht das Überwachen bestimmter Metriken, welche entweder schon vorgegeben sind, oder noch selbst geschrieben werden müssen. Dazu gibt es ein Webfrontend, das aus den gesammelten Informationen verschiedene Grafiken erstellt, um auf einen Blick sehen zu können, welche Systeme ordnungsgemäß funktionieren und bei welchen es Probleme gibt. Nagios kann an eine SQL-Datenbank angebunden werden und lässt einem so die Möglichkeit die Daten unabhängig auszuwerten.
Es ist kein Maximum an zu überwachende Hosts angegeben, sodass man testen muss, inwieweit Nagios skaliert und es für den speziellen Fall in Frage kommt. [6] [7]

CheckMK ist eine Erweiterung zu Nagios und in der Version *Check_MK Raw Edition* kostenlos erhältlich. Laut Angaben des Entwicklers existieren Installationen, welche Daten von bis zu 100.000 Checks pro Minute auf einem Server verarbeiten können. Allerdings gibt es keine Angaben zu einer maximalen Anzahl an zu überwachenden Hosts. Die Möglichkeit eigene Metriken zu definieren, welche überwacht werden sollen, ist auch bei *Check_MK Raw Edition* gegeben, sodass eine Erweiterung mit eigenen Metriken kein Problem darstellt. [14] Gesammelte Daten werden in sogenannten *Round Robin Archives*, kurz *RRA*, gespeichert, welche mit dem RRDtool erstellt werden. Dabei

ändert sich die Größe eines solchen Archives nicht. Sobald neue Daten nachkommen, welche keinen freien Platz mehr haben, werden die ältesten Daten überschrieben. Über die Dichte der Datensätze in einem Archiv bestimmt die *Konsolidierungsfunktion*, wie zum Beispiel der Mittelwert. Für jede gewünschte Datendichte gibt es ein eigenes Archiv, sodass am Ende zum Beispiel eines existiert, das für 10 Tage alle Datensätze enthält, eines welches für 10 Monate je einen Tagesmittelwert speichert und noch eines, das über 5 Jahre je einen Monatsmittelwert enthält. Über den Befehl `rrdtool dump` kann der Inhalt eines angegebenen Archivs als XML-Datei exportiert und auch ergänzt werden. Mittels `rrdtool restore` kann diese XML-Datei wieder in ein Archiv eingefügt werden. [15]

Munin Ist ein Monitoringtool um vernetzte Ressourcen zu überwachen. Munin legt sein Hauptaugenmerk auf die Darstellung von Trends, sodass die Möglichkeit besteht, festzustellen, warum die Performanz zu einem bestimmten Zeitpunkt eingebrochen ist. Munin wirbt zudem mit der Fähigkeit zu „Plug and Play“. Das bedeutet, es können im laufenden Betrieb neue zu überwachende Hosts angeschlossen werden. Munin speichert die ermittelten Werte für das Monitoring in einem RRA, wie auch CheckMK. Das bedeutet, dass auch hier der Umweg über den Export in eine XML-Datei gemacht werden muss, um die Daten weiter verarbeiten, beziehungsweise eigene Daten hinzufügen zu können. Munin bezieht die Daten von den Hosts über Nodes, welche einzelne oder mehrere Plugins beaufsichtigen. Die Plugins sind einfache Skripte mit einem standardisierten Output. Dabei ist die Skriptsprache egal, sodass auch sehr schnell und unkompliziert neue, benutzerdefinierte Plugins geschrieben werden können, um eigene Metriken umzusetzen. [17]

CollectD Ist ein kostenloses, open-source Monitoring-Programm, welches rein in der Programmiersprache C geschrieben ist, sodass es minimalen Performance-Einbußen gerecht wird. Dies hat den Vorteil, dass es auf „EmbeddedSystems“betrieben werden kann, die keine Skriptsprache unterstützen. Aufgrund effizienter Ressourcennutzung, können sowohl ein paar hundert, als auch mehrere tausend Hosts überwacht werden. Die anfallenden Daten dazu werden auch hier in RRA abgelegt und können über einen XML-Dump exportiert werden. Zusätzlich gibt es auch für CollectD die Möglichkeit das System Nagios zu integrieren. [10]

Ganglia Ganglia ist ein sehr bekanntes Monitoring-Programm um Clusterverbände zu kontrollieren. Es nutzt weit verbreitete Technologien wie XML für die Datendarstellung, XDR für den kompakten, portablen Datentransport, sowie RRDtool für die Datenspeicherung und -visualisierung. Wie auch bei Munin und CheckMK werden die

Daten in einem RRA gespeichert. Seine Datenstrukturen und Algorithmen sorgen für sehr niedrige Kosten pro Knoten und hohe Parallelität. Mit Ganglia ist ein Clusterverbund von bis zu 2000 Knoten möglich und kann auf allen gängigen Betriebssystemen zum Einsatz kommen. Um für Ganglia neue Metriken zu definieren gibt es das Tool *gmetric*. Wer sich in dieses nicht einarbeiten möchte, hat die Möglichkeit, Nagios zu integrieren und darüber neue Metriken zu erstellen. [13]

DataCenter-DataBase Aufgrund der räumlichen Nähe zum Leibnitz-Rechenzentrum, dem IT-Dienstleister der Münchner Universitäten und Hochschulen, bietet es sich an, auch das dort eingesetzte Monitoringsystem zu evaluieren. DataCenter-DataBase, kurz DCDB, ist eine Eigenentwicklung des LRZ und ist auf das Monitoring von Clustern ausgerichtet. Das definieren eigener Metriken ist hier nicht nur möglich, es ist sogar erforderlich. Die bestehenden Konfigurationsdateien enthalten Beispiele für zu überwachende Metriken, welche auf das eigene Computersystem angepasst werden müssen. An die überwachten Werte gelangt man auch direkt über das Interface der *cqlsh*. Diese ermöglicht es Operationen auf der Datenbank auszuführen und manuell Daten hinzuzufügen oder zu exportieren.

Zabbix Auch Zabbix bietet die Möglichkeit mit sogenannten *User-Parametern* eigene Metriken, welche überwacht werden sollen, zu definieren. Diese Definition beinhaltet ein Tupel aus einem unikaten Namen und dem Pfad zu einem ausführbaren Shellskript, welches den Wert zurückgibt, der kontrolliert werden soll. Diese User-Parameter können dann, wie die vorgefertigten Metriken direkt als Item einem zu kontrollierenden Host zugewiesen werden. Als Datenbank kann Zabbix mit verschiedenen Datenbankmanagementsystemen betrieben werden. Dazu zählen unter anderem PostgreSQL, MariaDB und MySQL. Bei diesen und auch den anderen zur Auswahl stehenden Datenbanken, besteht immer die Möglichkeit die Daten direkt einzusehen, sowie auch Daten zu exportieren oder einzufügen. Zabbix gibt es als kostenfreies Monitoringsystem und soll mehrere tausend Geräte überwachen können. [8] [9]

2.2 Bewertung der Monitoringsysteme

Wie im vorherigen Kapitel beschrieben werden die Systeme anhand der Kriterien Erweiterbarkeit, Skalierbarkeit und der Möglichkeit Daten zu exportieren, sowie der Verfügbarkeit einer kostenfreien Variante bewertet.
Dazu findet sich in der Abbildung 2.1 eine Übersicht der in Kapitel 2.1 genannten Systeme. Hieraus ist ersichtlich, dass für den geforderten Einsatzzweck PARMON nicht eingesetzt werden kann, da es keine Möglichkeit gibt eigene Metriken zu definieren

Kriterium	PARMON	Nagios	CheckMK	Munin	CollectD	Ganglia	DCDB	Zabbbix
benutzerdefinierte Metriken	nein	Exec Plugins	Exec Plugins	Skript mit standardi-sierter Ausgabe	bedingt	gmetric	über conf-Dateien	UserParam eter
Skalierbarkeit	ja	N/A	ja	N/A	ja	ja	ja	ja
Datenbank	eigene db-Dateien	MySQL-Datenimport über NDOUtils	RRA	RRA	RRA	RRA	Cassandra	SQL
Kosten	nein	nein	nein	nein	nein	nein	nein	nein

Abbildung 2.1: Kurzübersicht der Systeme aus Kaptiel 2.1

und auch die Roh-Daten aus dem Monitoring nicht exportiert werden können. Es besteht lediglich die Möglichkeit, sich die Daten anzeigen zu lassen. Nach Rücksprache mit dem Lehrstuhl soll auch kein System eingesetzt werden, das die Daten in RRA speichert, da hier kein direkter Zugriff auf die Monitoringdaten möglich ist. So bleiben noch Nagios, DCDB und Zabbix als mögliche Monitoringsysteme zu prüfen. Bei Nagios ist es ebenso aufwändig, wie mit den RRA, sodass auch dieses nicht weiter betrachtet wird. Im Folgenden werden die beiden verbleibenden Systeme genauer beschrieben.

Der Aufbau von **DCDB** entspricht dem gängigen Schema von Monitoringsystemen. Es gibt eine Komponente, welche lokal die Daten einsammelt und dann an eine zentrale Stelle (Server) schickt. Dort werden die Daten verarbeitet und in einer Datenbank gespeichert. In diesem Fall gibt es einen sogenannten *Pusher*, welcher die Daten über das Protokoll MQTT, Message Queue Telemetry Transport verschickt. Damit diese richtig zugeordnet werden können, ist jede zu überwachende Metrik mit einer eindeutigen ID versehen, welche auch gleichzeitig als *topic*-Referenz für das Übertragungsprotokoll verwendet wird. Auf dem Monitoringserver werden die verschickten Daten von einem *Collectagent* eingesammelt und in die Datenbank geschrieben.
Interessant ist dabei die Übertragung der Daten per MQTT-Protokoll. Diese ist darauf ausgelegt möglichst wenig Bandbreite für die Übertragung der Daten zu beanspruchen. Möglich wird das über den Aufbau nach dem *publish and subscribe*-Prinzip, sodass nur die gesammelten Daten übertragen werden, ohne bei jedem Sendevorgang erneut eine Verbindung aufbauen zu müssen.[1]

Zabbix ist ebenso nach diesem Prinzip aufgebaut. Der *Zabbix-Agent* sammelt die erforderlichen Informationen auf den zu überwachenden Hosts ein und schickt sie dann an den Server.
Die Daten werden in der Monitoringdatenbank gespeichert und über ein Webinterface, über welches ebenso die administrative Konfiguration erfolgt, grafisch aufbereitet.

3 Installation

3.1 Installation der Monitoringprogramme

Da die rein theoretischen Aspekte kein klares Ergebnis liefern, wird ein Vergleich der Systembeeinträchtigungen herangezogen. Dafür müssen die beiden Systeme, Zabbbix und DCDB, installiert werden, sodass äquivalente Tests ein eindeutiges Entscheidungskriterium bringen.

3.1.1 Zabbix

In der Dokumentation zu Zabbix sind als Systemvoraussetzungen 128 MB Arbeitsspeicher, sowie anfangs 256 MB Festplattenspeicher angegeben. Dieser wird sich je nach Länge der Parameterhistorie und der Anzahl an zu überwachenden Hosts verändern. In der Dokumentation wird außerdem angegeben, dass es mehrere Gigabyte sein können, die belegt werden.
An Software wird für Zabbix ein Datenbankmanagementsystemen benötigt. Wobei frei zwischen MySQL, MariaDB, PostgreSQL, Oracle, IBM DB2, SQLite und InnoDB gewählt werden kann. In Absprache mit dem Lehrstuhl und aufgrund persönlicher Erfahrung, fiel die Entscheidung auf eine Installation mit MariaDB. Im Vergleich zu MySQL bietet dies folgende Vorteile:
Es stehen mehr Befehle zur Verfügung, wie zum Beispiel *FLUSH* oder *SHOW*, um die Datenbank an sich besser administrieren zu können. Des Weiteren gibt es die Möglichkeit, sich einen Fortschritt in % anzeigen zu lassen. Gerade für ein *ALTER TABLE*, welches oft sehr lange dauert, ist dies sehr nützlich.
Bei den Datentypen hat man mit MariaDB den Vorteil, dass auch Mikrosekunden unterstützt werden. [2]

Für die Installation der Datenbank wird zunächst überprüft, welche Versionen von MariaDB über den *Package Manager* zu bekommen sind. Dies wird über die *tab-Vervollständigung* im Terminal mittels `apt-get install mariadb-` erreicht. Im Mai 2018 ist die Version MariaDB 10.3.7 die letzte Stabile. Diese ist auf der *vmschulz12*, kurz VM, noch nicht verfügbar, sodass der *Package Manager* um die aktuelle Version nachgerüstet werden muss. Dazu bietet die Webseite der MariaDB Foundation eine Hilfeseite im Download-Bereich, mit der die entsprechenden Befehle in Abbildung 3.1 zur Verfügung

gestellt werden.

```
$ sudo apt-get install software-properties-common
$ sudo apt-key adv --recv-keys --keyserver hkp://keyserver.ubuntu.com:80
      0xF1656F24C74CD1D8
$ sudo add-apt-repository 'deb [arch=amd64,arm64,i386,ppc64el]
      http://mirror2.hs-esslingen.de/mariadb/repo/10.3/ubuntu xenial main'
```

Abbildung 3.1: Laden des aktuellen Repository für MariaDB

Die Installation der Datenbank kann dann wie in Abbildung 3.2 durchgeführt werden. Die Routine der „secure installation "prüft die Einstellungen der Datenbank mittels der in Abbildung 3.3 gezeigten Abfragen.

```
$ sudo apt-get update
$ sudo apt-get install mariadb-server
$ sudo mysql_secure_installation
```

Abbildung 3.2: Installation der Datenbank MariaDB

```
Change the root password? [Y/n] n
Remove anonymous users? [Y/n] Y
Disallow root login remotely? [Y/n] Y
Remove test database and access to it? [Y/n] Y
Reload privilege tables now? [Y/n] Y
```

Abbildung 3.3: Sicherheitsabfragen zu den Datenbankeinstellungen von MariaDB

Nun ist die Installation von MariaDB abgeschlossen und die Datenbank für das Monitoring kann eingerichtet werden. Dazu wird mittels `mysql -u root -p` die Verbindung zur Datenbank hergestellt und wie in Abbildung 3.4 eine neue Datenbank, sowie Benutzer erstellt:

Nachdem die Datenbank aufgesetzt ist, kann mit den weiteren Vorbereitungen zur Installation von Zabbix fortgefahren werden. Dazu gehört eine Apache Webserver Installtion, damit das Zabbix Frontend gehostet werden kann. Zabbbix benötigt dafür die Version 1.3.12 oder höher.
Des Weiteren wird php 5.4.0 oder höher mit den folgenden Erweiterungen voraus-

```
mariadb > CREATE DATABASE monitoring;
mariadb > CREATE USER 'zabbix'@'localhost' IDENTIFIED BY 'password';
```

Abbildung 3.4: Einrichten der Datenbank für das Monitoring unter MariaDB

gesetzt: gd, bcmath, ctype, libXML, xmlreader, xmlwriter, session, sockets, mbstring, gettext, ldap, ibm_db2, mysqli, oci8, pgsql und sqlite3
Um im Servermonitoring auf die verschiedenen Sockets zugreifen zu können, sind das OpenIPMI, libssh2, fping, libcurl, libiksemel, libxml2 und net-snmp.
Weiterhin stellte sich im Installationsprozess für die Testumgebung heraus, dass die Pakete libevent-dev, libcurl4-gnutls-dev, libpcre3-dev und libcurlpp-dev benötigt werden.

3.1.2 DataCenter DataBase

Die Installation von DCDB ist an sich sehr einfach aufgebaut. Da es jedoch noch keine ausführliche Dokumentation gibt, ist die nachfolgende Vorgehensweise nur eine Abschrift der Vorgänge, welche durchgeführt wurden. Eine Garantie auf Vollständigkeit und dass diese Vorgehensweise auch auf anderen Distributionen und Netzwerkumgebungen funktioniert, kann daher nicht gegeben werden.
Als Voraussetzung für eine erfolgreiche Installation hat sich folgendes ergeben:

1. Für den sogenannten *Collectagent* half das Paket libboost-all-dev bei der Installation.
2. Damit der sogenannte *Pusher* installiert werden kann, hat sich herausgestellt, dass nur das Paket libboost-log-dev zu benötigt wird.
3. Um für das Monitoring Daten über IPMI abfragen zu können, muss dieses vorher installiert werden. Dazu werden der Reihe nach die Befehle in Abbildung 3.5 ausgeführt.

```
$ wget http://ftp.gnu.org/gnu/freeipmi/freeipmi-VERSION.tar.gz
$ tar xzvf freeipmi-VERSION.tar.gz
$ cd freeipmi-VERSION
$ sudo apt-get install build-essential libgcrypt11-dev
$ ./configure
$ make
$ sudo make install
$ sudo ldconfig
```

Abbildung 3.5: Befehlsübersicht freeipmi

Danach kann mit der eigentlichen Installation begonnen werden, welche in einem Verzeichnis stattfindet, das im Folgenden mit DCDB-Home bezeichnet wird. In diesem Verzeichnis sollte der Benutzer Lese-, Schreib- und Ausführrechte besitzt. Zuerst werden die Daten des git-Repository in das DCDB-Home-Verzeichnis geladen. Anschließend werden die Abhängigkeiten geladen und der Installtionsprozess gestartet. Die auszuführenden Befehle sind in Abbildung 3.6 angegeben.

```
User@Host:DCDB-Home$ git clone https://gitlab.lrz.de/dcdb/dcdb.git
User@Host:DCDB-Home$ cd DCDB-Home/dcdb
User@Host:DCDB-Home/dcdb$ make deps
User@Host:DCDB-Home/dcdb$ make install
```

Abbildung 3.6: Befehlsübersicht DCDB Serverinstallation

Für den DCDB-Pusher ist der Vorgang kürzer. Hierzu genügt es das entsprechende git-Repository in das Verzeichnis DCDB-Home zu laden und danach den Installationsprozess zu starten. Abbildung 3.7 zeigt die auszuführenden Befehle.

```
User@Host:DCDB-Home$ git clone https://gitlab.lrz.de/dcdb/dcdbpusher.git
User@Host:DCDB-Home$ cd DCDB-Home/dcdbpusher
User@Host:DCDB-Home/dcdbpusher$ make
```

Abbildung 3.7: Befehlsübersicht DCDBpusher

Um das Monitoring zu starten, wird zuerst die Datenbank verknüpft und danach der Collectagent gestartet. Dieser läuft mit dem Parameter „d “im Hintergrund. Als Default-Einstellungen werden jeweils die localhost-Adresse für Datenbank und MQTT-Broker verwendet, sowie der Port 1883 für MQTT und 9042 für die Datenbank. Username und Passwort sind für die Monitoringdatenbank nicht gesetzt. Die Terminalcommands dazu sind in Abbildung 3.8 zu sehen.

```
User@Host:DCDB-Home/install/cassandra/bin$ ./cassandra
User@Host:DCDB-Home/install/bin$ ./collectagent -d
```

Abbildung 3.8: DCDB Startup

3.2 Aufgetretene Probleme

3.2.1 Zabbix

Nach einem Neustart des Serves kann es sein, dass sich der Service zabbix-agent nicht starten lässt. Das Problem dabei ist, dass in das sogenannte *pid-file* nicht geschrieben werden kann, welches die Identifikationsnummer des Prozesses festhält. Um den Service dennoch wieder starten zu können, muss sichergestellt werden, dass es im Verzeichnis `/run` den Ordner *zabbix* gibt und dieser als Besitzer den Benutzer *zabbix* eingetragen hat. Danach kann der Service wieder mit `sudo service zabbix-agent start` gestartet werden.

3.2.2 DataCenter DataBase

Anfangs waren nur ein paar Informationen zur Architektur von DCDB bekannt und es gibt noch keine komplette Dokumentation. Daher wurde davon ausgegangen, dass ein Broker für MQTT, sowie die Datenbank Cassandra zusätzlich installiert werden müssen, was zu teils unnötigen Fehlern führte.
Beim ersten Versuch den Collectagent mit der separat eingerichteten Datenbank zu starten, gab es Meldungen, welche darauf hindeuteten, dass ein Boost-Paket fehle. Nach Installation des Paketes *libboost-all-dev*, trat ein weiterer Fehler auf, dessen Logmeldung nicht aussagekräftig genug war, um eine Lösung dazu zu finden. Durch Gespräche mit LRZ-Mitarbeitern wurde bekannt, dass die Datenbank zum einen in der Installationsroutine mit aufgesetzt und mit in Abbildung 3.8 genanntem Befehl `cassandra` verknüpft wird.
Ähnliches gilt für das Übertragungsprotokoll MQTT. Dieses wird über den Messaging-Server Mosquitto realisiert und ebenfalls in der Installationsroutine mit umgesetzt. Der Broker muss allerdings nicht separat gestartet werden.

4 Evaluierung

4.1 Versuchsaufbau

Um die Tests durchführen zu können, stehen insgesamt 4 Maschinen zur Verfügung. Diese sind in Tabelle 4.1 mit ihrer Spezifikation aufgelistet.

Info	vmschulz12	i10se1	i10se4	i10se5
CPU	Intel(R) Xeon(R) CPU E5-2697A v4	Intel Xeon Silver 4116 CPU	Intel Xeon Silver 4116 CPU	Intel Xeon Bronze 3106 CPU
Taktfrequenz	2,6	2,1	2,1	1,7
RAM in GB	4	98	98	82
/-Speicher in GB	77	886	886	900

Tabelle 4.1: Auflistung der Hardware für die Messungen

Die Serverkomponenten von Zabbix und DCDB sind auf der vmschulz12 installiert. Auf der i10se1, i10se4 und i10se5, im Folgenden Hosts genannt, ist jeweils der Zabbix-Agent, sowie der DCDB-Pusher eingerichtet.
Um den Einfluss von DCDB und Zabbix auf das System messen zu können, werden je ein Test zur Messung der Prozessorauslastung, dem Verbrauch an RandomAccessMemory (RAM) und zur Ermittlung der Temperatur der einzelnen Prozessorkerne erstellt. Es erfolgen je drei Messungen auf den Hosts. Eine, bei der keine Komponente der Monitoringprogramme läuft, eine mit laufendem Zabbix-Agenten, sowie eine mit laufendem DCDB-Pusher. Während der Prüfung sind alle anderen vom Monitoringprogramm zur Verfügung gestellten Tests abgeschalten. Es wird jede Messung mit einer Dauer von zwei Tagen ausgeführt.

4.2 Ergebnisse

Die Messungen des genutzten Arbeitsspeichers sind in der Abbildung 4.1 zu sehen. Dort sind drei Diagramme zu sehen, die jeweils die Messergebnisse mit und ohne laufenden Monitoring-Agenten zeigen. Die Abszisse zeigt das jeweilige Datum inklusive Uhrzeit an und die Ordinate die gemessenen Werte genutzten Arbeitsspeichers. Damit der optische Vergleich möglich ist, ist der Bereich der Ordinatenbeschriftung identisch. Analog dazu finden sich in Abbildung 4.2 die Werte der jeweils noch verfügbaren Kapazität an Prozessorleistung. Auch hier gibt die Abszisse die Zeit an, während auf der Ordinate die noch freie Kapazität an Prozessorleistung aufgetragen ist.

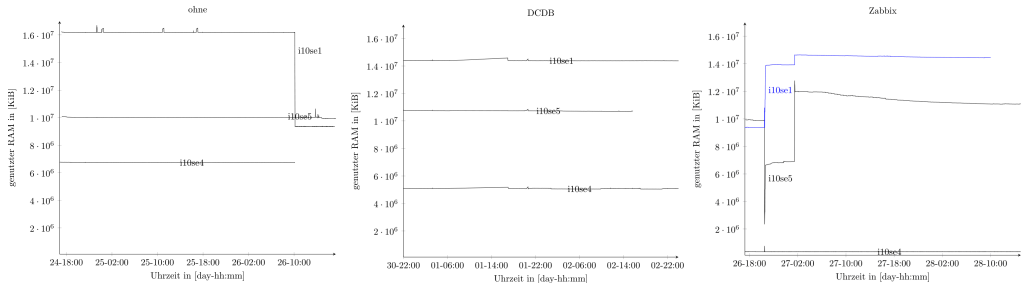

Abbildung 4.1: Vergleich der genutzten Kapazität an RAM der Hosts i10se[1,4,5]

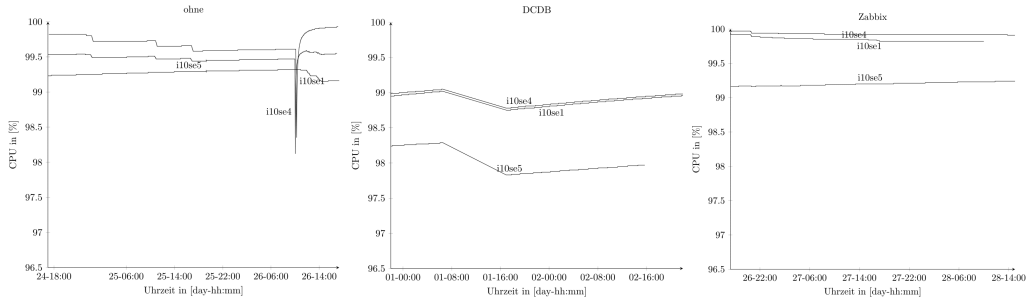

Abbildung 4.2: Vergleich der freien CPU-Kapazität der Hosts i10se[1,4,5]

Des Weiteren wurde eine Messung mit dem Benchmarkingtool LULESH gemacht, welches unter anderem von der LLNL entwickelt wurde. LULESH misst die Zeit, die ein System benötigt, um eine kubische Berechnung durchzuführen. [12]
Mit dem Befehl `openmpi -np 8 lulesh2.0 -i 10` wird diese Berechnung zehn Mal hintereinander ausgeführt und auf acht Kerne verteilt.
Auch hier wieder je eine Ausführung ohne die Agents, eine mit Zabbix und eine mit

DCDB. Die Ergebnisse dazu sind in den Tabellen 4.2,4.3 und 4.4 dargestellt.

Durchgang	ohne Agenten	mit DCDB	mit Zabbix
1	126,2	126,35	126,3
2	126,28	126,84	126,68
3	126,55	126,59	126,65
4	126,42	126,76	126,69
5	126,31	126,49	126,35

Tabelle 4.2: Ausführungsdauer des Programm „lulesh2 “in Sekunden auf der i10se1

Durchgang	ohne Agenten	mit DCDB	mit Zabbix
1	169,51	168,32	168,76
2	169,11	168,66	168,89
3	168,89	169,27	169,55
4	168,86	168,47	168,26
5	168,83	168,29	168,9

Tabelle 4.3: Ausführungsdauer des Programm „lulesh2 “in Sekunden auf der i10se4

Durchgang	ohne Agenten	mit DCDB	mit Zabbix
1	389,16	413,77	368,74
2	365,4	387,11	376,27
3	375,78	398,47	390,43
4	371,1	388,9	396,55
5	400,89	402,63	376,34

Tabelle 4.4: Ausführungsdauer des Programm „lulesh2 “in Sekunden auf der i10se5

Für eine bessere Vergleichbarkeit wurde aus diesen Werten jeweils ein Mittelwert berechnet, welche in Tabelle 4.5 eingetragen sind.

In Abbildung 4.3 ist dargestellt, inwieweit sich die gemessenen Werte des Arbeitsspeicher zu einem Median zusammenfassen lassen. Diese Mittelwerte sind in Tabelle 4.6 nochmals in Zahlen zusammengefasst. Daraus geht hervor, dass sich die Arbeitsspeichernutzung auf der i10se1 und der i10se4 auf einen höheren Wert einpendelt, wenn keine Agents laufen. Bei der i10se5 hingegen wird unter laufenden Agents mehr Arbeitsspeicher genutzt, als ohne. Dies entspricht der Intuition, lässt aber zusammen mit den Ergebnissen der beiden anderen Maschinen keinen Schluss zu, welches Programm

	i10se1	i10se4	i10se5
ohne laufender Agenten	126,35	169,04	380,47
mit DCDB-Pusher	126,61	168,6	398,18
mit Zabbix-Agent	126.53	168,87	381,67

Tabelle 4.5: Ausführungsdauer des Programm „lulesh2 “im Durchschnitt in Sekunden

die höheren Performanzeinschnitte herbeiführt.

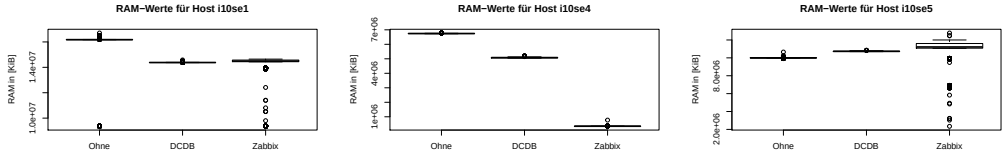

Abbildung 4.3: Vergleich Medianwerte für die Arbeitsspeichernutzung der Hosts i10se[1,4,5]

	i10se1	i10se4	i10se5
ohne laufender Agenten	15.195.136	6.744.648	10.024.374
mit DCDB-Pusher	14.380.700	5.077.300	10.832.720
mit Zabbix-Agent	14.031.341	346.640	10.805.262

Tabelle 4.6: Mittelwerte der Arbeitsspeichernutzung in [KiB]

Analog zu den Abbildungen und Tabellen der Arbeitsspeichernutzung, sind auch die Werte der freien CPU-Kapazitäten in Abbildung 4.4 und Tabelle 4.7 aufbereitet. Eine Tendenz zu stärkeren Performanzeinschnitten eines der beiden Monitoring-Agenten ist nicht zu erkennen. Die Medianwerte unterscheiden sich gesamt betrachtet um maximal 0,7%, wobei auch hier teilweise niedrigere Werte mit laufenden Agents vorhanden sind, als im Vergleich zur Messung ohne.

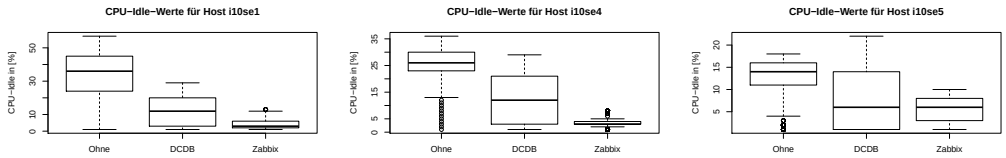

Abbildung 4.4: Vergleich Medianwerte der nicht genutzten CPU-Kapazität der Hosts i10se[1,4,5]

	i10se1	i10se4	i10se5
ohne laufender Agenten	15.195.136	6.744.648	10.024.374
mit DCDB-Pusher	14.380.700	5.077.300	10.832.720
mit Zabbix-Agent	14.031.341	346.640	10.805.262

Tabelle 4.7: Mittelwerte der nicht genutzten CPU-Kapazität in [%]

4.3 Auswirkung

Im Hinblick auf den verbrauchten Arbeitsspeicher ist nicht ersichtlich, ob DCDB oder Zabbix besser abschneidet, da die Messergebnisse nicht eindeutig sind. Dies kann man an den unterschiedlichen Höhen der Graphen aus Abbildung 4.1 sehen. Insbesondere liegt der Graph für den Durchlauf ohne Agenten bei Host i10se4 höher, als bei der Durchführung mit laufendem DCDB-Pusher oder Zabbix-Agenten. Für die Hosts i10se1 und i10se5 ist eine Aussage zum Vergleich ebenso schwierig, da es hier starke Anstiege und Abfälle im Verbrauch gibt.
Anhand der gemessenen Werte der noch freien Prozessorkapazität ist man geneigt, den Vergleich zwischen den beiden Programmen zu ziehen, wonach der Zabbix-Agent deutlich weniger Ressourcen einzunehmen scheint. Die Werte aus der Messung ohne laufender Agenten jedoch lässt diese Aussage nicht zu, sodass auch hier kein Favorit ermittelt werden kann.
Aus zeittechnischen Gründen konnten von den Messungen mit „lulesh2 “jeweils nur fünf Ausführungen durchgezogen werden. Tabelle 4.5 zeigt für die Hosts i10se1 und i10se4 Abweichungen im Bereich von hundertstel Sekunden. Dabei ist die Tendenz auf der i10se1, dass der Zabbix-Agent die Ausführung minimal weniger verzögert, als der DCDB-Pusher. Hingegen auf der i10se4 dauert im Durchschnitt von fünf Ausführungen die Messung mit laufenden Agents weniger lang, als ohne, also sogar eine positive Abweichung. Auf der i10se5 ergab die Messung wieder eine negative Abweichung. Hier von über sechzehn Sekunden mehr unter laufendem DCDB-Pusher, als unter dem Zabbix-Agent. Die Abweichung der Ausführungsdauer unter laufendem Zabbix-Agent

beträgt im Vergleich zur Messung ohne Agents dagegen nur knapp über 1 Sekunde.

4.4 Empfehlung

Alle Ergebnisse zusammengenommen, lässt sich folgern, dass durch den Zabbix-Agent und den DCDB-Pusher das System nur minimal beeinträchtigt wird und der Einsatz der beiden Monitoringprogramme gleichermaßen berechtigt ist. Deshalb kann nur eine differenzierte Empfehlung gegeben werden. Möchte man eine ausführliche Dokumentation zu dem einzusetzenden Monitoringprogramm zur Verfügung haben, so ist es ratsam Zabbix zu verwenden. Komplettiert wird diese noch durch das Zabbix-Forum, hinter der eine große *Community* steckt, welche bei Fragen jeglicher Art weiterhilft. Hingegen ist DCDB zu empfehlen, wenn das Monitoringprogramm dem Standard des IoT entsprechen soll.

5 Fazit und Ausblick

Abschließend ist also zu sagen, dass beide Monitoringsysteme ihrer Vor- und Nachteile haben. Es bringt zwar durchaus Vorteile, das System DCDB mit einer no-SQL-Datenbank (Cassandra) zu verwenden, allerdings ist dies keine Out-Of-The-Box-Lösung, sodass es zum Teil sehr langwierig ist, alle Build-Fehler in den Griff zu bekommen, bis überhaupt ein Grundsystem läuft. Das liegt vor allem daran, dass es bisher nur eine kurze ReadMe-Datei gibt, welche grundsätzlich beschreibt, wie für den DCDB-Pusher vorzugehen ist. Dies erfordert jedoch einen lauffähigen Collectagent. Die wichtigen Informationen zu Systemvoraussetzungen und Abhängigkeiten zu anderen Programmen sind leider nirgends definiert und müssen im Installationsprozess herausgefunden werden. Des Weiteren gibt es keinen offiziellen Ansprechpartner, da das System nur für den Eigenbedarf entwickelt wurde.
Zabbix hingegen bietet deutlich mehr Hilfe bei der Installation und Konfiguration des Monitoringsystems über eine offizielle Dokumentation und ein eigenes Forum. Was in dieser Arbeit nicht mehr untersucht werden konnte, ist das Hinzufügen von Daten, welche unter anderem für Voraussagen zusätzlich wichtig sind und der Umgang des Monitoringsystems mit diesen „Fremddaten“. Mit der Monitoring-Umgebung selbst werden nur alle diejenigen Daten erfasst, welche direkt vom System abgefragt werden können. Möchte man nun aber weitergehen und nicht nur auf entstandene Probleme reagieren, sondern schon präventiv entgegenwirken können, so ist es erforderlich, dass auch Daten zu äußeren Einflüssen mit Zeitstempel gesammelt werden. Das schließt zum Beispiel die Raumtemperatur des Serverraumes mit ein, oder Wettereinflüsse, sowie auch Schwankungen im Stromnetz.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] A. Banks und R. Gupta. „MQTT Version 3.1. 1“. In: *OASIS standard* 29 (2014).

[2] D. Bartholomew. „MariaDB vs. MySQL“. In: *Dostopano* 7.10 (2012), S. 2014.

[3] Bibliographisches Institut GmbH, Hrsg. *Duden - Die deutsche Rechtschreibung*. `https://www.duden.de/rechtschreibung/Monitoring`. [Online - aufgerufen am 04.06.2018 um 18:27]. 2017.

[4] Bibliographisches Institut GmbH, Hrsg. *Duden - Die deutsche Rechtschreibung*. `https://www.duden.de/rechtschreibung/Cluster`. [Online - aufgerufen am 25.06.2018 um 12:34]. 2017.

[5] R. Buyya. „PARMON: a portable and scalable monitoring system for clusters“. In: *Software-Practice and Experience* 30.7 (2000), S. 723–740.

[6] Ethan Galstad. *Datenbank Unterstützung*. `http://nagios.sourceforge.net/download/contrib/documentation/german/1_0/xdata-db.html`. [Online - Aufgerufen am 25.06.2018 um 18:52].

[7] Ethan Galstad. *Datenbank Unterstützung*. `https://library.nagios.com/library/products/nagios-core/`. [Online - Aufgerufen am 25.06.2018 um 18:12].

[8] *Features*. `https://www.zabbix.com`. [Online - aufgerufen am 6.7.2018 um 11:14].

[9] *Features*. `https://www.zabbix.com/documentation/3.0/manual`. [Online - aufgerufen am 6.7.2018 um 11:14].

[10] F. Forster. *Features*. `http://collectd.org/features.shtml`. [Online - aufgerufen am 5.7.2018 um 16:49].

[11] A. Hanemann, J. W. Boote, E. L. Boyd, J. Durand, L. Kudarimoti, R. Łapacz, D. M. Swany, S. Trocha und J. Zurawski. „Perfsonar: A service oriented architecture for multi-domain network monitoring“. In: *International conference on service-oriented computing*. Springer. 2005, S. 241–254.

[12] L. L. N. Laboratory. „LULESH“. In: (2014).

[13] M. L. Massie, B. N. Chun und D. E. Culler. „The ganglia distributed monitoring system: design, implementation, and experience“. In: *Parallel Computing* 30.7 (2004), S. 817–840.

[14] Matthias Kettner. *Check_MK*. `https://mathias-kettner.de/check_mk.html`. [Online - Aufgerufen am 26.06.2018 um 10:59].

[15] T. Oetiker. *rrdtool Documentation*. 2003.

[16] A. Vrenios. *Linux cluster architecture*. Sams, 2002.

[17] *Welcome to the Munin Guide*. `https://munin.readthedocs.io/en/latest/`. [Online - aufgerufen am 26.06.2018 um 18:49].