Fachbereich 2
Informatik und Ingenieurwissenschaften

Bachelorarbeit

Aufbau eines Clusters mit der NoSQL-Datenbank MongoDB auf Basis von Einplatinencomputern

Im Studiengang Informatik

zur Erlangung des akademischen Grades
Bachelor of Science

Thema:	Aufbau eines Clusters mit der NoSQL-Datenbank MongoDB auf Basis von Einplatinencomputern
Autor:	Danijel Klaic <dklaic@gmx.de> MatNr. 996907
Version vom:	16. August 2016
1. Betreuer:	Prof. Dr. Baun
2. Betreuer:	Prof. Dr. Gabel

Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

$Unterschrift:$ $Ort, Datum:$

Danksagung

Ganz besonders möchte ich Herrn Prof. Dr. Baun herzlich danken für seine fachliche und persönliche Betreuung, sowie für die Bereitstellung der technischen Mittel im Rahmen meiner Bachelorarbeit. Auch möchte ich mich bei meinem Zweitkorrektor Prof. Dr. Gabel für die Annahme meiner Bachelorarbeit bedanken.

Weiterhin bedanke ich mich bei meinen Eltern und Geschwistern, die mir durch ihre Unterstützung mein Studium ermöglicht haben. Mein Dank gilt ebenfalls meinen Freunden für die fachliche Ratschläge und moralische Unterstützung im Laufe meiner gesamten Studienzeit.

Zusammenfassung

Ziel war es in der vorliegenden Bachelorarbeit, einen eigenen Cluster mit der dokumentenorientierten NoSQL-Datenbank MongoDB aufzubauen auf Basis von dem Einplatinencomputer Raspberry Pi. Die Server (Knoten) vom Cluster sollten mit derselben Datenbank korrespondieren. Raspberry Pi eignet sich für die Analyse und für den Aufbau eines eigenen Cluster. Der Zusammenschluss mehrerer Geräte (Raspis) dieses Typs erfolgte über ein Netzwerk.

Dabei wurden die verteilten Funktionalitäten, wie die Ausfallsicherheit und Partitionierung der Daten (Sklaierbarkeit) von der MongoDB untersucht. Des weiteren wurden Vergleichungen von den Aggregationsmethoden durchgeführt. Diese Bachelorarbeit ist sowohl für Informatiker als auch für Leute, die sich mit Datenbanktechnologien auseinandersetzen interessant.

Abstract

The main goal of this thesis was to built up a cluster, which is based on a document-oriented database called MongoDB and a single board computer Raspberry Pi. The nodes of the cluster should use the same database. Raspberry Pi is suitable to analyze and compose a cluster. The composition of several devices (Raspis) was done via a network.

The following functionalties investigated Reliabitlity and Data partitioning on MongoDB. Additionally comparisons were performed by aggregation methods. This thesis is interesting for computer scientists as well as for people who deal with database technologies.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Raspis	Mehrere Geräte von Raspberry Pi
SEQUEL	Structured English QUEred Language
SQL	Structured Query Language
DDL	Data Definition Language
IBM	International Business Machines Corporation
GPS	Global Positioning System
NoSQL	Not Only Structured Query Language
MongoDB	Mongo Datenbank
SWOT	Strengths, Weaknesses, Opportunities, Threats
BASE	Basically, Available, Soft State, Eventual consistency
FIFO	First In - First Out
SD	Secure Digital
USB	Universal Serial BUS
SSH	Secure Shell
GPIOS	General-purpose input/output
CRUD	Create, Read, Update, Delete
JSON	JavaScript Object Notation
BSON	Binary JSON
ID	Identifikator
CPU	Central Processing Unit
GPU	Graphics Processing Unit
DHCP	Dynamic Host Configuration Protocol
IP	Internetprotokoll
APT	Advanced Packaging Tool
TCP	Transmission Control Protocol
HTTP	Hypertext Transfer Protocol

1 Einleitung

Edgar Frank Codd gelingt es in den 1960er und 1970er Jahren ein großer Fortschritt mit der Entwicklung von den ersten relationalen Datenbanksystemen [1]. In dieser Zeit haben relationale Datenbanken immer mehr an Bedeutung gewonnen. So entstand auch in den 1970er auch die Datenbanksprache SEQUEL (**S**tructured **E**nglish **QUE**red **L**anguage) basierend auf den relationalen Datenbanken. Später wurde jedoch die Sprache unbenannt in SQL (Structured Query Language) [2].

SQL und relationale Datenbanken arbeiten nach festen Tabellenschemen. Einer der Sprachelemente vom SQL ist Data Definition Language (DDL). DDL gibt vor, zuerst die Tabelle zu erstellen und nachträglich gibt es die Möglichkeit auch Spalten und Datentypen zu erzeugen [3]. Die Flexibilität von dem oben aufgeführten Schema ist begrenzt. Die Menge der Datenmenge ist über die Zeit stetig gewachsen. Laut IBM entstehen jeden Tag um die 2,5 Trillionen Byte an Daten. Das heißt, dass 90 % der Daten aus den letzten 2 Jahren entstanden sind. Die großen Datenmengen entstehen aus allen Bereichen, wie zum Beispiel aus dem Internetbereich, GPS-Technik und Sensortechnik [4]. Dies führt oft dazu, dass Datenbankmanagementsysteme und Hardware an ihre Grenzen stoßen. Bewältigung dieser großen Datenmengen stellten für Entwickler neue Herausforderungen. Somit entstanden neue NoSQL-Technologien. Dies war ein Einstieg für die Entwicklung von Google MapReduce und Hadoop von Yahoo [5]. Diese damals neuen Technologien verarbeiteten besser die großen Datenmengen als zuvor, somit war es ein wichtiger Fortschritt [5]. Die NoSQL-Welt spielt heute wesentlich eine große Rolle für die Datenverarbeitung von Big Data. Somit ist der Begriff NoSQL nicht mehr wegzudenken aus der Datenbankwelt.

NoSQL ist eine Abkürzung für "Not only SQL" ("nicht immer nur SQL"). Dies bedeutet nicht, dass man auf SQL bzw. relationaler Datenbank verzichtet, sondern auch Alternativen einbezieht zur SQL-Welt [6]. NoSQL-Implementierungen sind in mehrere Kategorien unterteilt [6], die in dieser Bachelorarbeit näher erklärt werden. Meistens werden große Datenmengen schneller verarbeitet als im Gegensatz zu relationalen Datenbanken. Immer mehr NoSQL-Datenbanken verbreiten sich. Zurzeit ist die NoSQL-Datenbank MongoDB am beliebtesten und bei allen Datenbanken auf dem vierten Platz [7]. Aus diesem Grund fiel die Entscheidung auf NoSQL-Datenbank MongoDB. Es handelt hierbei um eine quelloffene Software. MongoDB verwendet für die Implementierung der Datenbank, die Programmiersprache C++. [8]. Zudem gibt es zahlreiche Mongo-DB-Treiber, die in den Programmiersprachen: C, C++, Java, Ruby, Phyton und PHP zu Verfügung stehen.

Viele bekannte Großunternehmen wie Facebook, Google, Expedia und Bosch benutzen zurzeit MongoDB [9].

Der Begriff Mongo leitet sich vom englischen Wort "hu**mongo**us" was übersetzt bedeutet "gigantisch" oder "wahnsinnig groß", es wird auf den Volumen Aspekt von Big Data hingedeutet [10]. Zurzeit besteht keine exakte Definition für Big Data. Das Softwareunternehmen 10gen entwickelte im Jahr 2007, die erste Version von MongoDB. Im Jahre 2013 kam es auch zu einer Namensänderung des Softwareunternehmens, die sich ebenfalls in MongoDB umbenannte. Neben dem Volumen Aspekt sind noch zwei weitere Velocity und Variety Aspekte vorhanden. Besonders für die Big-Data-Aspekte Volume und Velocity empfiehlt sich die Verwendung von verteilten Systemen. Der Begriff Big Data entstand im Jahr 2001 von Doug Laney (damaligen Analysten der Meta Group). Laney definierte Big Data als "Daten mit großem Volumen ("Volume"), großer Vielfalt an Datenformaten ("Variety") und großen Geschwindigkeit ("Velocity") mit der neue Daten entstehen" [11].

1.1 Aufbau

Die Bachelorarbeit gliedert sich in sechs Teilen. Auf die Einleitung folgend, widmet sich der zweite Teil mit der Entstehung und den Grundlagen von Raspberry Pi und MongoDB. Weiterhin befasst sich das dritte Kapitel mit dem Aufbau, Installation, Konfiguration und den dazugehörigen Hardware- und Softwarekomponenten.

Im Mittelpunkt des vierten Kapitels steht die Implementierung der Datenreplikation und Datenfragmentierung innerhalb eines Clusters. Es werden hier die Funktionalitäten eines Cluster ausgeführt und ausgewertet.

Das Kapitel 5 befasst sich mit den Aggregationsmethoden. Der Schlussteil dieser Arbeit endet mit einem Fazit.

2 Technische Grundlagen

Das zweite Kapitel gibt einen Einblick in die Grundlagen von NoSQL-Datenbanken und Raspberry Pi.

2.1 NoSQL-Definition

Bislang gibt es keine einheitliche und strikte Definition von NoSQL. Den Versuch einer Definition haben die Autoren Edlich, Friedland, Hampe, Brauer in ihrem Buch [12] vorgenommen. Die besagt:

- **Kein relationales Datenmodell**
  Eine Vielfältigkeit der Modelle wird gefordert, dies bedeutet, dass das relationale Modell nicht immer für jede Anforderung am besten geeignet ist. Für Projekte eignet sich die SWOT-Analyse (Strengths/Stärken, Weakness/Schwächen, Opportunities/Chancen, Threats/Bedrohungen), die analysiert welches Modell bzw. Datenbanksystem infrage kommt [12].

- **Eignung für Systeme mit verteilter und horizontaler Skalierbarkeit**
  Relationale Datenbanksysteme arbeiten in der Regel nachdem Scale-up Prinzip bzw. vertikale Skalierbarkeit, in dem eine technische Aufrüstung des Rechners stattfindet. Im Gegensatz zu den relationalen Datenbanksysteme arbeiten die NoSQL-Datenbanksysteme oft nachdem Scale-out-Prinzip bzw. horizontale Skalierbarkeit, die eine schnellere Reaktionszeit auf Anfragen und Manipulationen durch dynamisches Einbinden und Löschen von Rechnerknoten es ermöglichen. Aufgrund einer vorhandenen Fehlertoleranz ist auch im vorliegenden Fall ein Knotenfall ohne Konsequenz möglich. Dadurch sinken die Anschaffungskosten [12].

- **Quelloffene Software**
  Die meisten NoSQL-Datenbanksysteme sind quelloffen und können dadurch eine Voraussetzung sein für eine SWOT-Analyse aber mit Sicherheit kein entscheidendes Argument [12].

- **Frei von Schemen oder nur von schwächeren Restriktionen betroffen**
  Die Agilitäten der Datenstrukturen von Web2.0-Anwendungen müssen in der heutigen Zeit steigen. Datenbanksysteme und Anwendungen müssen mit unterschiedlich strukturierten Informationen im Web- und Dokumentbereich umgehen können. Die Folge wäre, dass die Anwendung diese Schemainformation zurück verfolgen müsste. Die Voraussetzung sind schnellere Reaktionszeiten bei einer Schemaänderung. Schemaänderungen können nicht wie bei den relationalen Datenbanken, ein System minuten- oder stundenlang außer Betrieb nehmen [12].

- **Einfache Datenreplikation zur Unterstützung der verteilten Architektur**
  Die Grundidee bei einer Datenreplikation ist die Verteilung der Daten auf mehrere Knoten. Es wird eine schnelle und einfache Datenreplikation gefordert, damit die Daten auch bei einem Kontenausfall noch vorhanden sind. Die meisten NoSQL-Datenbanksysteme arbeiten damit erfolgreich. Diese Voraussetzung kommt zu Stande mit einer abgeschwächten Konsistenzanforderung, die in dem CAP-Theorem mit der Eigenschaft "evenutally consistency" vorhanden ist. Die Umsetzung erfolgt praktisch über das BASE-Konsistenzmodell [12].

- **Einfache API**
  SQL war sicherlich einfach, deklarativ, auch sehr ausdrucksstark und vor allem standardisiert. Der wesentlicher Nachteil bei vielen NoSQL-Datenbanksysteme ist, dass es zwar eine einfache API gibt aber nicht alle Funktionalitäten anbietet und somit nicht sehr bedeutsam. Dadurch entsteht die Folge, dass eine Entwicklung von Funktionalitäten für die Anwendung notwendig ist. Zur Zeit entstehen neue Funktionalitäten, die sich auch für Web2.0-Anwendungen eignen [12].

- **Kein ACID als Konsistenzmodell**
  Die ACID-Eigenschaften verkomplizieren und verlangsamen die verteilte Datenhaltung und stellen somit ein Problem für NoSQL-Datenbanksysteme da. Die Eigenschaft "evenutally consistency" ermöglicht den Ansatz, dass Daten in den verschiedenen Knoten vorübergehend auch inkonsistent sein dürfen. Dieses Szenario kommt oft in den Internet-Anwendungen vor und nicht zum Beispiel sehr oft bei klassischen Geschäftsdaten, in dem meistens relationale Datenbanksysteme infrage kommen. Es existieren auch Datenbanksysteme, die sowohl mit ACID als auch mit BASE arbeiten können [12].

2.2 CAP-Theorem

Ein wichtiger Aspekt der verteilten Datenbanksysteme ist das CAP-Theorem. Das CAP-Theorem besagt, dass nicht alle verteilten Eigenschaften, wie Konsistenz (Consistency), Verfügbarkeit (Availability) und Toleranz (Partition Tolerance) gleichzeitig zu gewährleisten sind. Somit können nur maximal zwei von drei Eigenschaften gleichzeitig vorhanden sein [13].

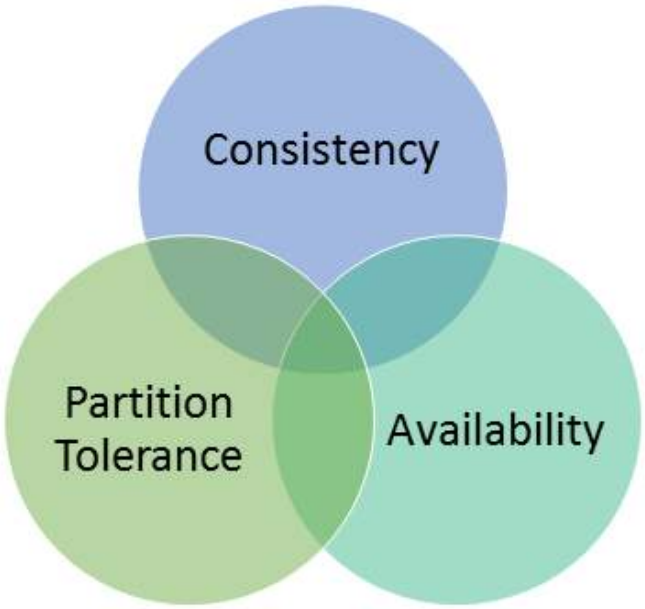

Abbildung 1: Veranschaulichung des CAP-Theorems

- **Consistency**:
  Alle Knoten verfügen gleichzeitig über die gleichen Daten [13].
- **Availbility**:
  Lese- und Schreibzugriffe sind jederzeit verfügbar. Ein Knotenausfall wirkt sich nicht aus auf die Verfügbarkeit [13].
- **Partition Tolerance**:
  Das System funktioniert auch bei einem Ausfall einzelner Knoten [13].

Die meisten NoSQL-Datenbanken können durch ihre hohe Verteilung, nicht auf Partition Tolerance verzichten [6]. MongoDB verwendet diese Eigenschaft durch replica sets [13]. Die zweite Eigenschaft von MongoDB ist die Consistency, dadurch ist das System nicht immer vorhanden, zum Beispiel bei einer Replikation aber die Daten bleiben konsistent. Bei der Eigenschaft Availability können die Daten unterschiedlich sein aber dafür sind sie immer verfügbar. Somit gehört MongoDB zur den CP-Kategorie. Relationale Datenbanksysteme zählen meistens zur der CA-Kategorie [6].

2.3 BASE

BASE steht für die Abkürzung **B**asically **A**vailable, **S**oft State, **E**ventual consistency und ist basierend auf NoSQL-Datenbanken. Es dient als Gegensatz zu ACID-Prinzip, dass für relationale Datenbanken basiert. BASE führt dazu, dass das verteilte System nach einer möglichst kurzen Zeitspanne der Inkonsistenz letztlich wieder in demselben konsistenten Zustand zurückführt. Ein Beispiel hierfür ist die Master-Slave-Replikation. Ein Knoten stellt den Master-Knoten und die anderen Knoten sind als Slave-Knoten zu betrachten. Der Master-Knoten repliziert Daten auf die anderen Slave-Knoten. Schreibzugriffe sind nur bei Master-Knoten vorhanden. Lesezugriffe sind bei allen Knoten vorhanden. Bei Datenänderung sind zunächst die veränderten Daten aus Sicht des Slave-Knoten nicht zu sehen. Erst nach der Vollendung dieser Replikation, sind die geänderten Daten für alle Knoten sichtbar. Der Ablauf verläuft asynchron [14].

2.4 NoSQL-Datenbanken-Vergleich

Bei den NoSQL-Datenbanken gibt es verschiedene Kategorien und Unterkategorien. In diesem Umfeld existieren verschiedene Anbieter und Konzepte. Folgende Kategorien sind bei NoSQL-Datenbanken bedeutend:

- **Spaltenorientierte Datenbanken (englisch Wide Column Stores)**:
  Daten werden durch Datensätze mit potenziell sehr vielen dynamischen Spalten gespeichert. Es besteht eine Ähnlichkeit mit Key-Value [6].

- **Dokumentenorientierte Datenbanken (englisch Document Stores Database)**:
  Die NoSQL-Datenbank MongoDB, die in dieser Bachelorarbeit zum Einsatz kommt, ist eine dokumentenorientierte Datenbank. Es handelt sich dabei nicht um typische Dokumente wie zum Beispiel Word oder PDF-Dokumente sondern von MongoDB JSON-Dokumente. JSON steht für die Abkürzung JavaScript Object Notation [13].

- **Key-Value-Datenbanken**:
  Key/Value-Datenbankysteme benutzen ein einfaches Schlüssel- und Wertschema. Zu jedem Wert ist ein bestimmter Schlüssel zugewiesen, der aus einer strukturierten oder beliebige Zeichenkette besteht. Ein Schlüssel ist in Namensräume sowie in Datenbanken aufgeteilt. Die Werte des Systems, die sich auch Values nennen, können neben Strings auch Listen, Sets oder Hashes enthalten [15].

- **Graphendatenbanken)**:
  Es handelt sich um eine Datenbank, die Graphen verwendet. Die Graphen bestehen aus Knoten und Kanten. Im Gegensatz zu den anderen NoSQL-Anbietern, erfolgt der Zugriff auf einen bestimmten Knoten oder in einer Menge von Knoten anstatt auf eine eindeutige ID [6].

Die nachfolgende Tabelle zeigt eine Auswahl von NoSQL-Anbieter einschließlich der zugeordneten Kategorien [16]:

Spaltenorientierte Datenbanken	Apache HBase, Cassandra, Amazon SimpleDB
Dokumentenorientierte Datenbanken	MongoDB, CouchDB, OrientDB
Key-Value-Datenbanken	Oracle NoSQL Database, DynamoDB, Redis
Graphdatenbanken	Neo4J, Sparksee, TITAN
Multi-Model-Datenbanken	ArangoDB, OrientDB, GunDB

Tabelle 1: NoSQL-Anbieter

2.5 MongoDB-Struktur

Folgende Abbildung zeigt den Aufbau einer MongoDB-Struktur Ein MongoDB-Server kann bis zu mehrere logische Datenbanken verwalten. Die Bearbeitung von Datenbanken kann individuell mit den Einstellungen separat erfolgen. Eine Datenbank bei MongoDB ist zum Beispiel aus folgenden Dateien aufgebaut:("Datenbankname.0, Datenbankname.1") und Namespace-File ("Datenbankname.ns") [8].

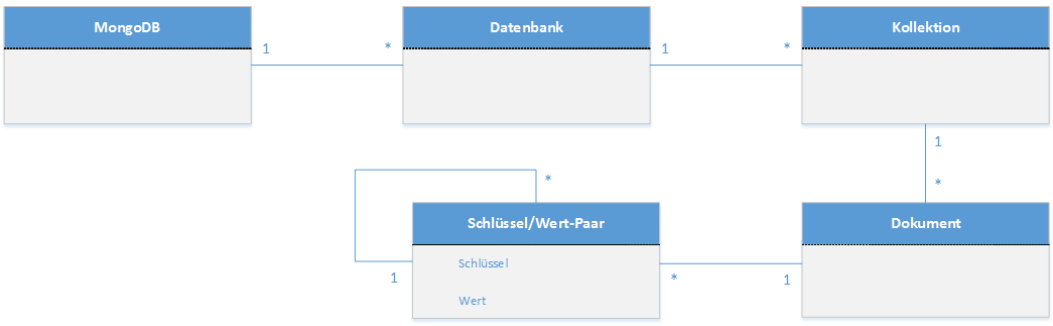

Abbildung 2: MongoDB-Struktur

2.5.1 BSON

Dokumente sind in BSON-Format enthalten. In einer Namespace-Datei sind alle Collections und Metadaten aufgelistet. BSON steht für die Abkürzung Binary JSON. Hierbei handelt sich um eine binäre Erweiterung von JSON [6]. Die Vorteile sind gegenüber JSON [6]:

- Bessere Reversierbarkeit
- Stabile Speicherung von bestimmten Datentypen
- Erweiterung für weitere Datentypen, wie zum Beispiel Integer- und Datumstypen

Ein JSON-Dokument kann beliebig viele Felder haben, denen ein Wert zugeordnet ist. Dabei kann der Wert mehrere Werte mit unterschiedlichen Datentypen erhalten, wie man im folgenden Beispiel sieht.

```
{
vorname: "Danijel"
nachname: "Klaic"
studiengang: "Informatik"
matrikelnummer: "996907"
}
```

Das JSON-Format eignet sich auch für komplexere Datentypen mittels Subdokumente. Bei Subdokumenten handelt es sich um beliebige Schachtelungen innerhalb eines Dokumentes, die es erlauben Unterattribute zu modellieren [13]. Das folgende Beispiel zeigt ein Subdokument *kontaktdaten* mit den zwei Feldern *handynummer* und *email*.

```
{
vorname: "Danijel"
nachname: "Klaic"
studiengang: "Informatik"
matrikelnummer: "996907"
}
kontaktdaten {
handynummer: "015784492739"
email: "dklaic@gmx.de"
}
```

2.5.2 ObjectID und Primärschlüssel

Jedes Dokument verfügt eine eindeutige Identifizierung mit dem Datentyp *ObjectID*, die auch als Primärschlüssel dient. Außerdem dient es auch zur Lokalisierung des Dokuments. Die Größe einer ObjectID beträgt zwölf Bytes und besteht aus 24 hexadezimalen Zeichen [6], wie im folgenden Beispiel:

Abbildung 3: ObjectID-Beispiel

Die ersten vier Bytes sind für den Zeitstempel reserviert, das die Sekunden seit 1.1.1970 auffasst. Die darauffolgenden drei Bytes leiten sich aus dem Maschinenidentifikationscode ab. Die nächsten zwei Bytes stehen für eine gegenwärtige Prozess-ID. Die letzten drei Bytes stehen für den Zähler und übernehmen einen Zufallswert. Es besteht auch die Möglichkeit, eine eigene ID zu erzeugen. Da jedes Dokument ein Feld besitzt mit dem _id, das für die Aufnahme des Primärschlüssels dient [13]. Im folgenden Beispiel zeigt die manuelle Eingabe einer selbst erzeugten ID:

```
db.studenten.insert(_id:1, vorname:"Danijel", nachname:"Klaic", studiengang:"Informatik", matrikelnummer:"996907")
```

Es empfiehlt sich bei großen Datenmengen die vom Treiber erzeugte ObjectID zu verwenden, da es unter Umständen zu Kollisionen führen kann.

2.5.3 Datentypen

JSON-Format unterstützt folgende Datentypen [13]:

Datentypen	**Beispiel**
ObjectID	Siehe Kapitel 2.2.2
String	Eine Reihe von Buchstaben, Zahlen und Sonderzeichen, die in Anführungszeichen abgeschlossen sind.
Integer, Double	Zahlen oder Kommazahlen, die in Anführungszeichen abgeschlossen sind.
Boolean	true oder false
Aray	Eine sortiere Liste von Werten mit beliebigen Datentypen.
Date	Zeigt ein Datum mit Uhrzeit und Zeitinformationen an. Das Kommando new.date() zeigt den aktuellen Zeitstempel.
Null	Der Wert null steht für ein reserviertes Feld. Soll das Feld leer sein, so erkennt der Benutzer, dass keine Eingabe erfolgte.

Tabelle 2: Datenypen

2.5.4 Collection

Eine Collection ähnelt der Tabelle einer relationalen Datenbank. Der wesentliche Unterschied ist dabei, dass die Dokumente keine technischen Voraussetzungen benötigen. Somit nennt man diesen Zustand "Schemafreiheit". Die Erstellung von Collections verläuft erst nach Bedarf. Durch eine Punktnotation erfolgt die Gruppen und Hierarchien Anordnung [6]:

Beispiel
Collection 1: europa.deutschland.hessen
Collection 2: europa.deutschland.bayern

Das obige Beispiel zeigt zwei unterschiedliche Collections. Der Namenspaces "europa.deutschland" steht für Zuordnung der Gruppe.

MongoDB unterscheidet zwischen zwei Formen von Collections [6]: **System Collections** und Capped **Collections**

Capped Collection ist eine Collection mit definierter Größe oder Anzahl der Dokumente. Sollte die jeweilige Grenze überschritten sein, durch das Einfügen von neuer Dokumenten, kommt das FIFO-Prinzip (First In - First Out) zum Einsatz und löscht die älteren Dokumente [6].

Folgende Tabelle zeigt Informationen an, die in einer **System Collections** enthalten sind [6].

Name	**Bedeutung**
system.indexex	Listet alle Indizes einer Datenbank auf, die in jedem einzelnen Collection angelegt sind.
system.namespaces	Hier befinden sich eine Liste mit allen Namespaces von den Datenbanken , die sich aus den Indizes herleiten.
system.profile	Diese Collection entsteht durch die Aktivierung des Profiling, in dem es mehr Informationen über die Datenbank sammelt.
system.users	Diese Collection entsteht durch die Aktiverung des Aufthentizerung. Hier befinden sich alle Informationen über den Benutzer.
system.roles	Listet die Definition von benutzerspezifischen Rollen.
system.js	JavaScript-Variablen und Funktionen können zentral abgelegt werden.

Tabelle 3: System Collections

2.5.5 CRUD

Bekanntlich benutzt MongoDB keine übliche Abfragesprache wie SQL, stattdessen kommt eine von JavaScript implementierte Abfragesprache zum Einsatz. Das JavaScript-Objekt *db* ist ein wichtiges Element und selektiert in der Mongo Shell die aktuelle Datenbank [6][17]. CRUD steht für die vier grundlegenden Datenbankoperationen und sind auch bei MongoDB enthalten. Folgende Tabelle zeigt die CRUD-Operationen:

Create	Datensätze erzeugen.	insert
Read	Datensätze lesen.	find
Update	Datensätze aktualisieren.	update
Delete	Datensätze löschen.	remove

Tabelle 4: CRUD-Operationen

Nachfolgender Text zeigt die Manipulation von Dokumenten, wie man Dokumente einfügt, ändert und löscht.

Create Das Einfügen von neuen Dokumenten erfolgt über die Kommando *insert()*. Als Parameter erwartet die Funktion ein BSON-Dokument, welches in die angegebene Collection eingefügt wird. Falls die Collection nicht existiert, wird diese angelegt [13].

```
>db.personen.insert(nachname:"Klaic", vorname:"Danijel")
```

Read Die Lesezugriffe erfolgen über das Kommando *find()*. Ohne Bekanntgabe des Parameters listet Kommando find() alle Dokumente der angegebenen Collection auf. Andernfalls nimmt der erste Parameter die Rolle des Selektors ein, während der zweite Parameter, die Ergebnisse wiedergibt [13].

```
>db.personen.find()
>db.personen.find(vorname:"Danijel")
```

Update Änderungen an den Dokumenten funktionieren über das Kommando update() *find*. Ähnlich wie beim find() Kommando ist der erste Parameter ein Selektor. Die Änderungen finden bei allen Dokumenten statt, die dem Muster des Selektors entsprechen und sich im zweiten Parameter befinden [13].

```
>db.personen.find()
>db.personen.find(vorname:"Danijel")
```

Delete Das Löschen der Dokumenten funktioniert über das Kommando *delete*. Das delete() Kommando funktioniert ähnlich wie das find() Kommando, nur ohne Projektion. Das heißt, es wird lediglich ein Selektor übergeben. Eine vollständige Löschung gibt es dem entsprechend, nur wenn die Kriterien des Selektors entsprechen [13].

```
>db.personen.reomove(vorname:"Danijel")
```

2.6 Raspberry Pi

Raspberry Pi ist ein Einplatinencomputer von dem britischen Unternehmen Raspberry Pi Foundation. Dieser Einplatinencomputer sollte es dazu bringen, in britischen Schulen größeres Interesse für das Programmieren zu wecken. Schnell expandierte der Einplatinencomputer weltweit und entwickelte sich zu einem großen Markterfolg [5]. Es ermöglicht ein Umfeld zum Experimentieren und Fehler zu machen ohne Verluste in Kauf nehmen zu müssen.

2.6.1 Raspberry Pi 3 Modell B

Für diese Bachelorarbeit kommt das Model Raspberry Pi 3 Modell B zum Einsatz. Im Gegensatz zu den Vorgängern bringt das neue Model mehr Leistung und eine Wireless Konitkivität mit sich. Die folgende Tabelle zeigt die technische Daten vom Raspberry Pi 3 Model B:

Prozessor	Broadcom BCM2837
CPU	1.2GHz 64-bit quad-core ARMv8
GPU	400 MHz VideoCore IV
Arbeitsspeicher	1 GB RAM LPDDR2-900 SDRAM
Kommunikationsarten	LAN 10/100 Mbps, 802.11n Wireless LAN und Bluetooth 4.1
Schnittstellen	4 x USB 2.0, 1 x HDMI, 1 x CSI Camera Port DSI Display Port, Stereo Out, microSD Port
Abmessungen	Maße 85,6 x 56,5 Millimeter
Gewicht	Ca. 45 Gramm

Tabelle 5: Technische Daten vom Raspberry Pi 3 Model B

Abbildung 4 zeigt einen Raspberry Pi 3 Modell B. Wie man auf dem Bild erkennen kann, verfügt der Raspberry Pi 40 GPIOS-Pins. Sie dienen für die Steuerung und Messung von technischen Geräten und spielen keine Rolle für diese Bachelorarbeit.

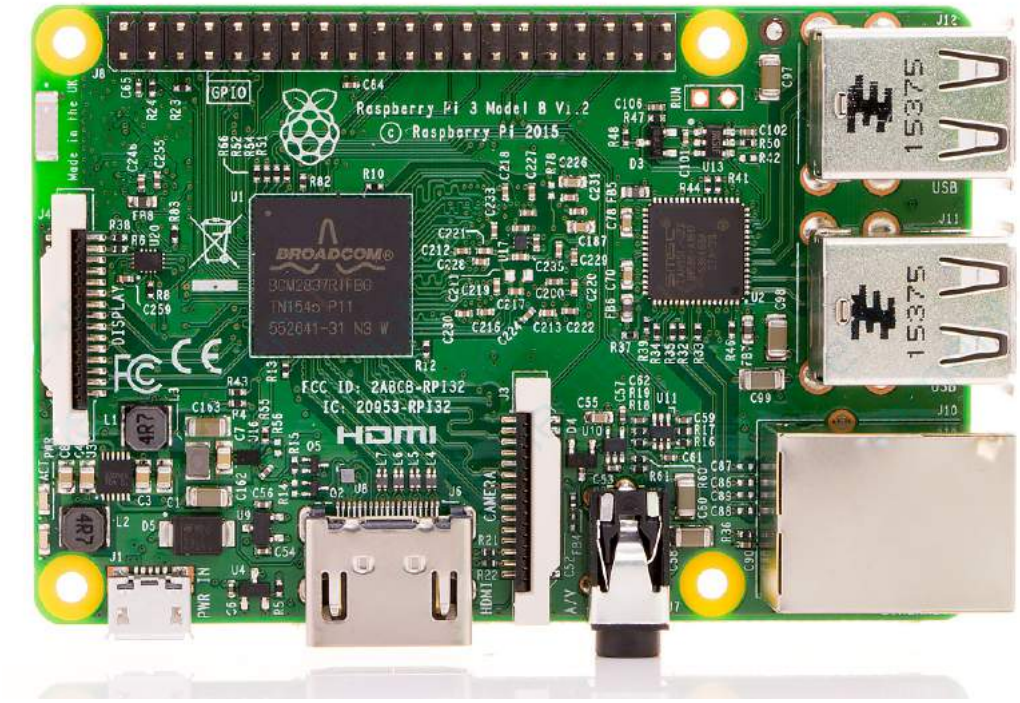

Abbildung 4: Raspberry Pi 3 Modell B[1]

[1]Bildquelle: https://javatutorial.net/wp-content/uploads/2016/03/RaspberryPi-3-Model-B.jpg

3 Aufbau und Installation eines Clusters mit der NoSQL-Datenbank MongoDB auf Basis von Raspberry Pi Einplatinencomputern

Das folgende Kapitel befasst sich mit dem Aufbau und der Installation eines Clusters mit der NoSQL-Datenbank MongoDB sowie mit den eingesetzten Hardware- und Softwarekomponenten.

3.1 Hardware- und Softwarekomponenten

Für die Durchführung dieser Bachelorarbeit kommen drei Raspis zum Einsatz. Weiterhin wird zusätzlich zu jedem Raspberry Pi eine SD-Speicherkarte eingelegt. Dies ist notwendig, um ein Betriebssystem auf den Raspberry Pi zu schreiben. Anstelle einer Festplatte nutzt der Raspberry Pi eine Micro-SD-Karte, wobei das Basisbetriebssystem etwa 2 Gigabyte auf der Karte belegt. Für vernünftiges Arbeiten empfiehlt sich eine Karte mit mindestens vier Gigabyte oder mehr mit der Geschwindigkeitsklasse [18].

Um die Stromversorgung zu entlasten, eignet sich hierfür ein USB-Hub. Neben den USB-Hub werden Netzwerkkabeln benötigt, um eine Verbindung zum Netzwerkanschluss herzustellen. Ein wichtiger Bestandteil ist der Router für die Verwendung von mehreren Netzknoten.

Raspberry Pi verfügt ein angepasstes Betriebssystem Namens Raspbian. Es ist eine Version von der Linuxdistribution Debian. Zum Installieren von Raspbian auf den Raspberry Pi wird zusätzlich ein weiterer Rechner benötigt, um die IMG-Datei vom Raspbian auf die Speicherkarte zu schreiben. Da der weitere Rechner unter dem Windows Betriebssystem lief, eignete sich dafür das Programm Win32DiskManager [18]. Win32DiskManager erstellt bootfähige SD-Karten für den Raspberry Pi.

Mithilfe von dem Windows Tool Putty hat man die Möglichkeit sich per Netzwerk durch SSH mit dem entfernten Raspberry Pi zu verbinden [18].Dadurch lässt sich die Eingabekonsole vom Raspbian per Windows Rechner steuern. Der Zugriff auf die Benutzeroberfläche vom Raspbian erfolgt durch die Remotedesktopverbindung von Windows.

Abbildung 6 zeigt die Hardwarekomponenten für diese Bachelorarbeit.

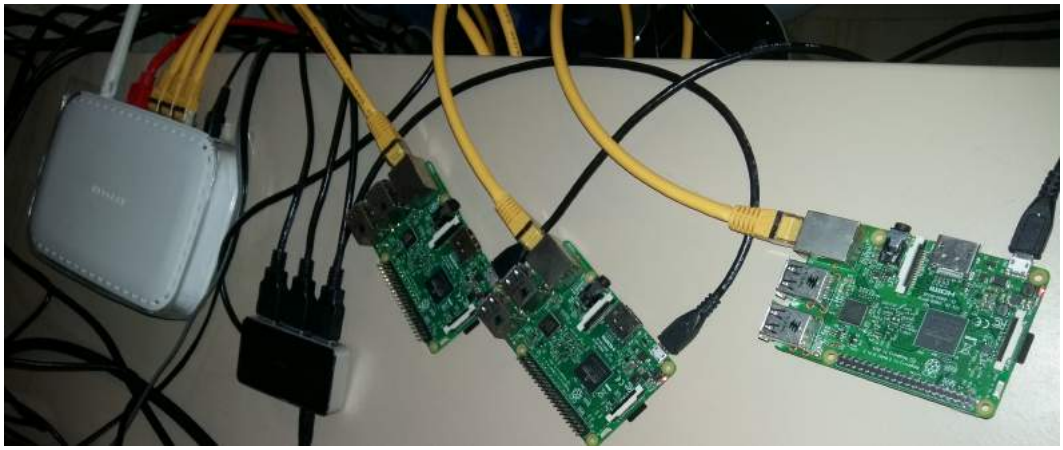

Abbildung 5: Aufbau eines Clusters

3.2 Installation und Konfiguration

Dieser Abschnitt zeigt in der Reihenfolge, die notwendigen Konfigurations- und Installationsschritte.

3.2.1 Raspberry Pi Installation

Für die Inbetriebnahme von Raspberry Pi, soll zunächst ein Betriebssystem auf der SD-Karte installiert werden. Die Betriebssysteme "Raspbian" und "Noobs" werden von Raspberry Pi Foundation als optimal gesehen und empfohlen [19]. Für diese Bachelorarbeit kommt derzeit die aktuellste Version "Raspbian Jessy" zum Einsatz. Der Start des Betriebssystems gelingt über die SD-Karte. Um es möglich zu machen, muss zunächst die Imagedatei vom Raspbian auf die SD-Karte geschrieben werden [18].

Nach dem Extrahieren, kann nun das Programm starten. Die folgende Abbildung zeigt die Benutzeroberfläche von Win32 Disk Manager. Um die Imagedatei auf die SD-Karte zuschreiben, muss zunächst der Pfad dieser Datei ausgewählt werden (siehe Abbildung 6).

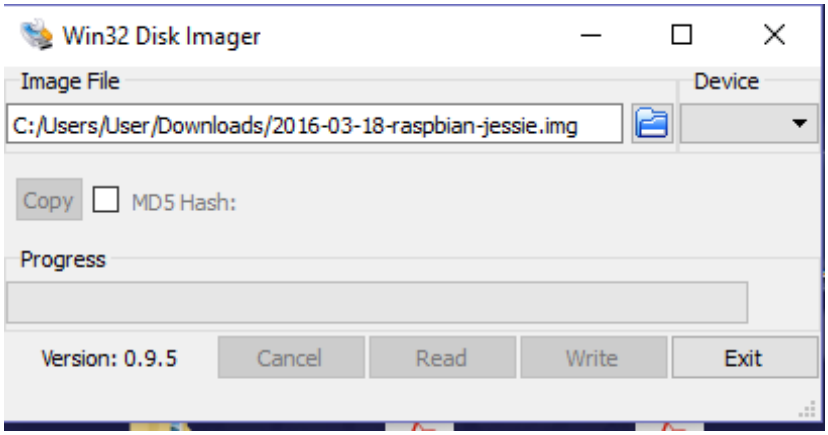

Abbildung 6: Win32 Disk Manager

3.2.2 Raspberry Pi Konfiguration

Nachdem die Raspis angeschlossen sind und der Zugriff über PuTTY erfolgte, taucht beim ersten Booten das Konfigurationstool raspi-config auf. Durch das folgende Kommando lässt sich das Tool auch manuell aufrufen:

```
sudo raspi-config
```

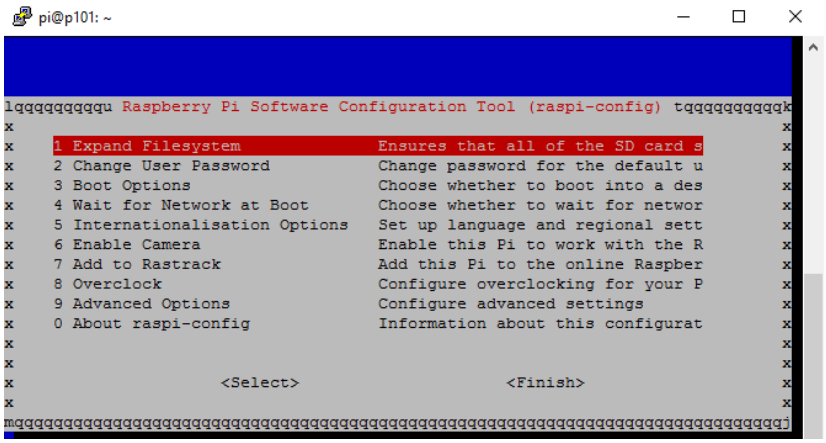

Abbildung 7: Menü des Konfigurations-Tool raspi-config

Abbildung 7 zeigt das Menü von raspi-config und deren Konfigurationsoptionen. Im Folgenden werden die wichtigsten Konfigurationsoptionen näher vorgestellt:

Expand Filesystem

Diese Option ermöglicht es, das Volumen der SD-Karte für Raspbian zu partitionieren für Raspbian. Die Standardsetzung gibt zwei GigaByte vor [20].

SSH

Diese Option ist besonders wichtig für die Fernsteuerung über das Netzwerk. Die Aktivierung ist standardmäßig und soll auch weiterhin bestehen bleiben [20].

Memory Split

Diese Option legt den Speicheranteil von CPU und GPU fest [20].

Overclock

Diese Option bestimmt die Geschwindigkeit des Prozessors [20].

Change Locale

Mit dieser Option lässt sich die Sprache und Zeichencodierung anpassen [20].

Nach der Anpassung dieser Konfigurationsoptionen soll ein Neustart erfolgen, um mit den neuen Einstellungen zu starten [20]. Dies passiert mit dem folgenden Kommando:

```
sudo shutdown -r 0
```

Nach dem Neustart erfolgt die Aktualisierung von den bereits vorhandenen Programmen, um den Raspberry Pi auf den aktuellen Stand zu bringen:

```
sudo apt-get update
sudo apt-get upgrade
```

Die Vergabe der IP-Adressen erfolgt automatisch durch den DHCP-Server vom Router. Um die IP-Adresse von den jeweiligen Geräten herauszufinden, kam das Programm Fing zum Einsatz. Wie auf der nachfolgenden Abbildung zu erkennen, sind die IP-Adresse "192.168.1.6", "192.168.1.7", "192.168.1.8" reserviert für die Raspis. Um sich eine bessere Übersicht zu verschaffen, erfolgt die Umbenennung der Hostnamen, wie folgt: p101, p201 und p301. Diese Änderung erfolgt über die Datei "*/etc/hostname"*:

```
sudo vi /etc/hostname
```

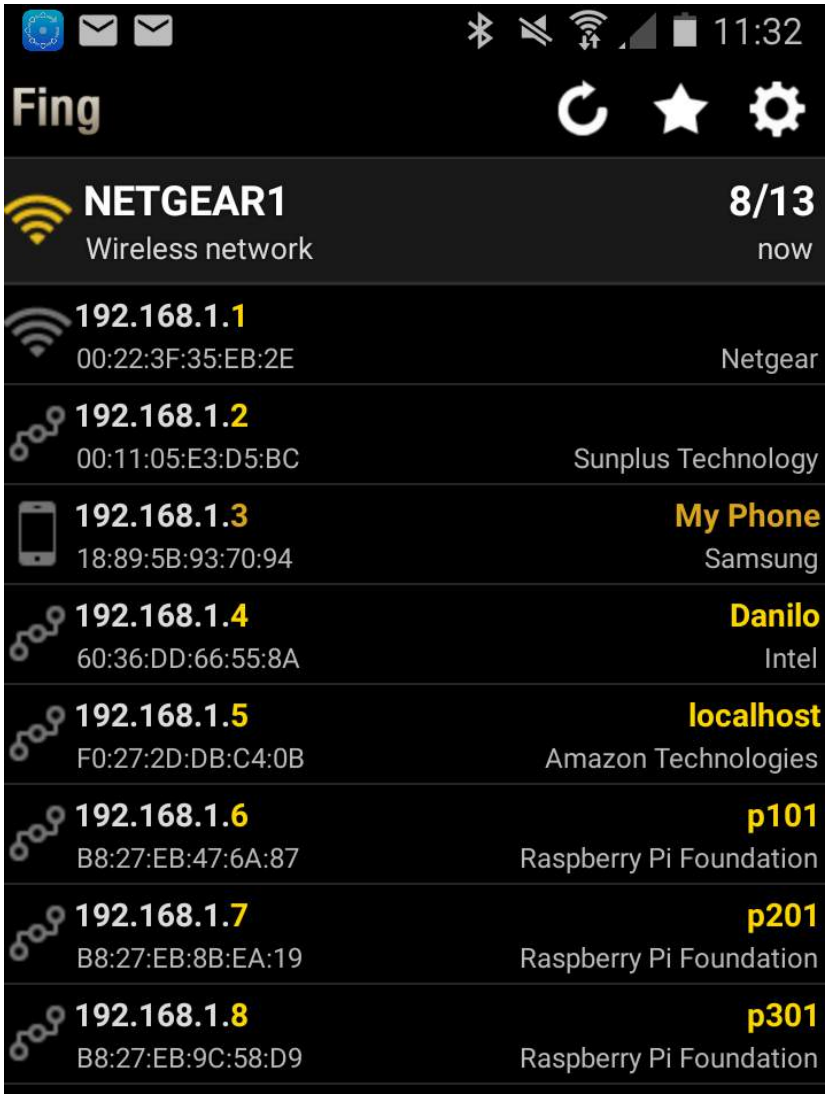

Abbildung 8: Programm Fing

Anstatt immer feste IP-Adresse zu verwenden, erfolgt eine automatische IPv4-Konfiguration per DHCP-Client Deamon (DHCPCD). Standardmäßig ist der DHCP Client Deamon bei Raspbian Jessie aktiviert. Die Konfiguration erfolge über die Datei "/etc/dhcpd.conf" [21].

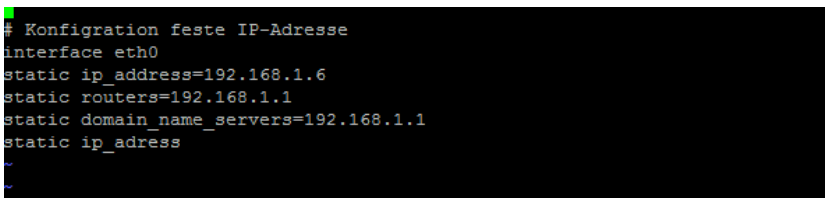

Abbildung 9: Datei "/etc/dhcpd.conf"

Für die PuTTY-Konfiguration ist es zunächst notwendig, die IP-Adresse oder den Hostnamen einzugeben und das SSH-Protokoll zu aktivieren. Benutzername "pi" und Passwort "raspberry" sind beim Betriebssystem Raspbian bereits standardmäßig vorgegeben. PuTTY benutzt die Portnummer 22 standardmäßig.

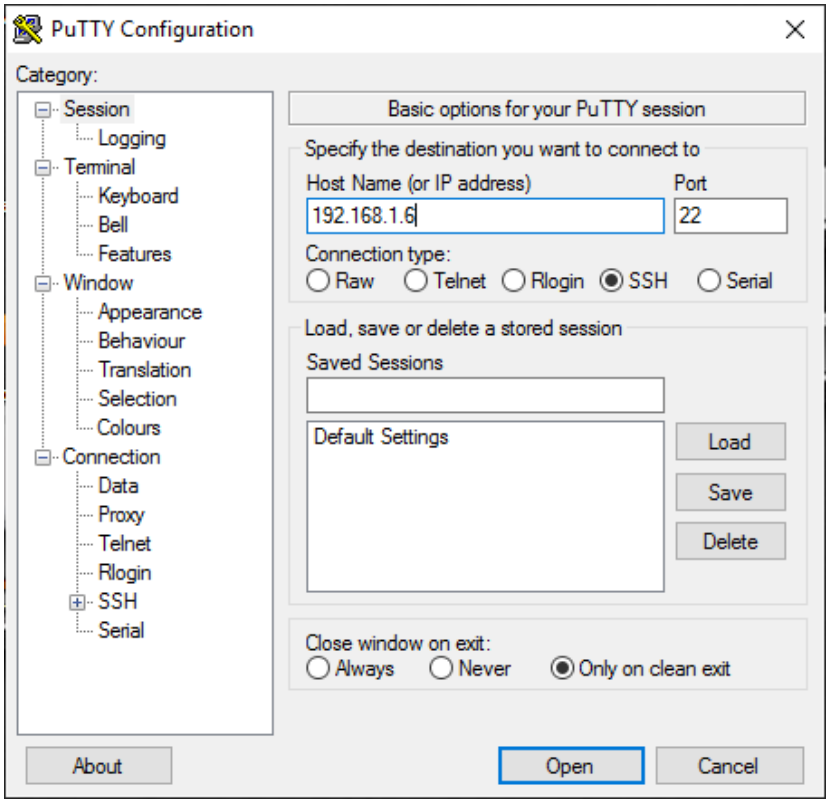

Abbildung 10: PuTTY

Da die Rasberry Pis von einem Windowsrechner gesteuert werden, erfolgt der Zugriff auf die grafische Oberfläche mit der Remotedesktopverbindung. Remotedesktopverbindung ist im Lieferumfang von Microsoft enthalten. Nun folgt die Installation des Remote Desktop Servers. XRDP ist eine freie Implementierung des Remote Desktop Protokolls. Diese können direkt mittels den Paketmanager APT installiert werden [22]. Nach Abschluss der Installation wird der XRDP-Daemon automatisch gestartet.

```
sudo apt-get install xrdp
```

Um eine Verbindung zu einen Remotedesktop herzustellen, kommt zunächst die Eingabe der IP-Adresse (siehe Abbildung 11).

Abbildung 11: Anmeldung per Remotedesktopverbindung

Nach der Hostname- und Passworteingabe, erscheint die grafische Oberfläche von Raspbian (Abbildung 12).

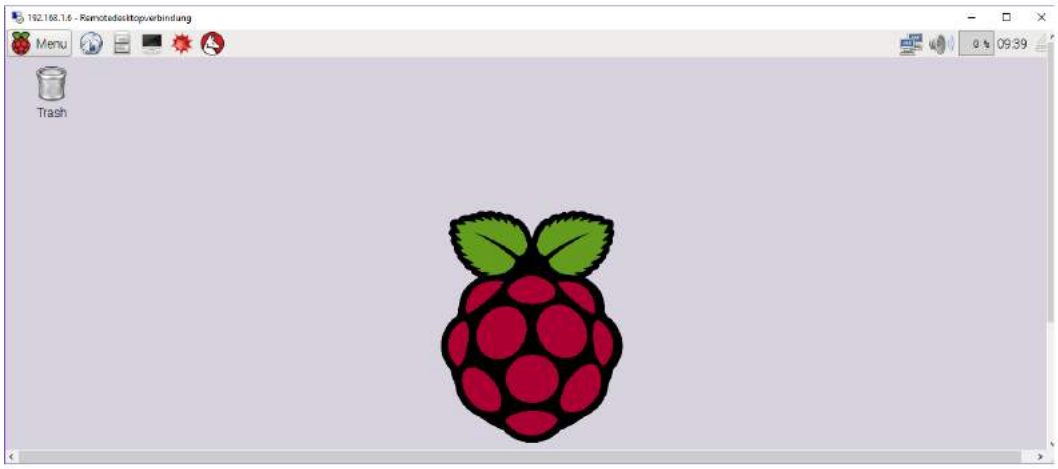

Abbildung 12: Grafische Oberfläche von Raspbian

3.2.3 MongoDB Installation

Als Erstes werden die Programme von MongoDB Version 3.0.9 heruntergeladen. Es handelt sich um eine 32-bit-Variante. Dies bedeutet, dass zwei Gigabyte zur Verfügung stehen. Die Installation erfolgt durch das Herunterladen einer gz-Datei, in dem die ausführbaren Dateien enthalten sind. Nach dem Extrahieren bedarf es keine weitere Installation.

```
wget https://www.andyfelong.com/downloads/core_mongodb.tar.gz
tar zxvf core_mongodb.tar.gz
```

Folgende Programme sind enthalten:

- **mongo**
  Die Mongoshell ist eine Kommandozeile, die direkt Befehle an MongoDB sendet [23].
- **mongod**
  Der Mongo Deamon ist ein Serverprozess und läuft auf einem bestimmten Port und nimmt Anfragen entgegen [24].
- **mongoperf**
  Repräsentiert die Festplatten-Leistung [25].
- **mongos**
  Mongos auch Sharding-Server genannt ist ein Router. Mongos verarbeitet die Abfragen und leitet diese an den Mongod-Server [26].

Weiterhin werden die Werkzeuge von MongoDB Version 3.0.9 heruntergeladen. Auch hier erfolgt die Installation und das Extrahieren einer gz-Datei.

```
wget https://www.andyfelong.com/downloads/tools_mongodb.tar.gz tar zxvf core_mongodb.tar.gz
```

Folgende Werkzeuge für MongoDB sind mit der jeweils dazugehörigen Funktionsbeschreibung enthalten:

- **mongoexport**
  Exportiert Daten von einer MongoDB Instanz, in einem JSON-Format [27].
- **mongoimport**
  Importiert Daten in einer MongoDB Instanz, in einem JSON-Format.
- **mongorestore**
  Erstellt einen Back-up in einer neuen MongoDB Instanz [28].
- **mongotop**
  Ausgabe der Ausführungszeiten von Lese- und Schreiboperationen von Collections [6].

- **mongodump**
  Erstellt Sicherungskopie von Daten [29].

- **mongofiles**
  Speichert und manipuliert die Daten von GridFS-Objekten [30].

- **mongooplog**
  Speichert alle schreibenden Operationen in einem Ringpuffer mit fester Größe. Dient als Grundlage für die Replikation [31].

- **mongostat**
  Periodische Ausgabe von Lese- und Schreibinformationen auf einer Datenbank und Collections [6].

Mit den folgenden Befehlen wird in das Verzeichnis von den Werkzeugen für MongoDB gewechselt. Benutzer werden zugewiesen und erhalten die für sie vorgesehene Berechtigung:

```
cd /home
sudo chown root:root mongo*
sudo chmod 755 mongo*
sudo strip mongo*
sudo cp -p mongo* /usr/bin
```

Hier wird ein Datenbankverzeichnis mit dem entsprechenden Benutzer und Berechtigungen erstellt [32]:

```
sudo mkdir /var/lib/mongodb
sudo chown mongodb:root /var/lib/mongodb
sudo chmod 775 /var/lib/mongodb
```

Hier wird ein Protokollverzeichnis mit dem entsprechenden Besitzer erstellt [32]:

```
sudo mkdir / var / log / mongodb
sudo chown mongodb:nogroup /var/log/mongodb
```

3.2.4 MongoDB Konfiguration

Die Konfigurationsdateien von MongoDB verwenden zur Serialisierung, die vereinfachte Auszeichnungssprache YAML. Die Erstellung von Konfigurationsdateien erleichtert die Verwendung von mongod und mongos Instanzen. Folgende Konfigurationsoptionen wurden verwendet:

- **port**
  Standardmäßig benutzt MongoDB den Port 27017. Es besteht auch die Möglichkeit, einen anderen freien Port zu benutzen [33].

- **shardsvr**
  Diese Kategorie wird für diese Arbeit eingesetzt und wird im Folgenden noch ausführlicher erläutert [33].

- **replset**
  Durch die Option replset wird der Replikationsname festgelegt [33].

- **quiet**
  Führt die mongod Instanz in einem ruhigen Modus und beendet die Aktivitäten [33].

- **dbpath**
  Bestimmung des Speicherortes für die Daten [33].

- **logpath**
  MongoDB erstellt eine Protokolldatei in dem angegebenen Pfad. Hier werden alle Diagnoseprotokollinformationen gesammelt [33].

- **logappend**
  Bei einem Neustart der mongod Instanz werden neue Einträge am Ende der vorhandenen Protokolldatei hinzugefügt. Ohne diese Option benutzt die mongod Instanz, die vorhandene Protokolldatei und erstellt eine neue Datei [33].

- **storageEngine**
  Der Systemdienst MMAPV1 übernimmt die Rolle für die Speicherengine. Es wird somit bestimmt, wie die Daten auf den Speicher und Festplatte geschrieben werden [6].

Der letzter Schritt ist die Erstellung einer MongoDB-Dienstdatei. Dabei kann der Dienst manuell bedient werden, um folgende Befehle auszuführen: Starten, neustarten, Status abfragen und beenden. Folgende Abbildung zeigt die MongoDB-Dienstdatei:

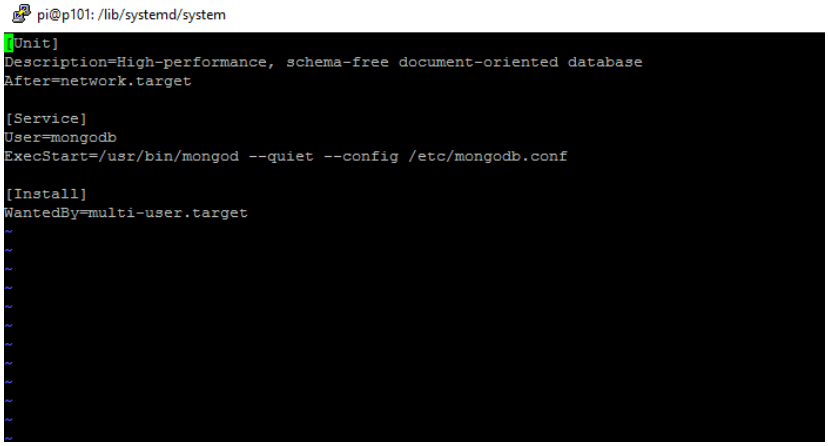

Abbildung 13: MongoDB-Dienstdatei

4 MongoDB

In diesem Kapitel werden die Grundlagen von Replication und Sharding näher vorgestellt. Anschließend erfolgt die Implementierung einer Replication- und Sharding-Umgebung in einem Cluster mit Raspis.

4.1 Grundlagen

In der Datei mongodb.conf befinden sich folgende Konfigurationseinstellungen.

dbpath	/var/lib/mongodb
logappend	true
storageEngine	mmapv1
quiet	true
rest	true
http	true

Tabelle 6: Konfigurationsdatei mongodb.conf

4.1.1 MongoDB-Dienst Starten

Um MongoDB zu bedienen, muss zunächst die Mongod-Instanz aktiviert werden. Dies passiert mit dem folgenden Befehl:

```
sudo mongodb start
```

Anschließend kann nun eine Verbindung mit Kommandozeilenprogramm Mongoshell hergestellt werden. Die Verbindung mit MongoDB-Server entsteht über die Protokolle TCP und IP. Mongo Shell wird mit dem folgenden Befehl aufgerufen:

```
mongo
```

Folgender Befehl steht für den Neustart der Mongod-Instanz:

```
sudo mongodb restart
```

Folgender Befehl steht für das Beenden der Mongod-Instanz:

```
sudo mongodb stop
```

4.2 Replikation

Die Ausfallsicherheit von den Daten wird durch die Replikation gesteigert. Dabei sind die replizierten Daten anhand von mehreren Knoten verteilt und somit zur weiteren Verwendung verfügbar. MongoDB unterscheidet zwischen zwei Arten von Replikation: Master-Slave-Replikation und Replica Sets [6].

4.2.1 Oplog

Für die Aktivierung dieser Replikation dient das Oplog (Abkürzung für Operations Log). In einem Oplog befinden sich alle schreibenden Operationen, wie die Befehle insert, update und delete. Das Feld op zeigt, welcher Operationstyp (i=insert, u=update, d=delete) enthalten ist. Das Feld ns übergibt die betreffende Collection [6].

4.2.2 Master-Slave-Replikation

Bei einer Master-Slave-Replikation haben die Instanzen eine feste Rolle. Eine Instanz übernimmt die Rolle des Masterknotens und somit haben weitere Instanzen die Rolle des Sekundärknotens. Der Masterknoten repliziert die Schreiboperationen an die Slaveknoten [6].

4.2.3 Replica Sets

Bei Replica Sets gibt es zwei Arten von Knoten: Primärknoten und Sekundärknoten. Primärknoten besitzen sowohl Lese- als auch Schreibrechte. Diese erstellen eine Kopie von den Daten auf die Sekundärknoten. Der Sekundärknoten verfügt über keine Lese- und Schreibrechte. Beim Ausfall eines Primärknoten wird ein Sekundärknoten zum neuem Primärknoten gewählt. Im Gegensatz zur Master-Slave-Replikation sind hier bei einzelnen Instanzen keine Feste Rollen vorgegeben [6].

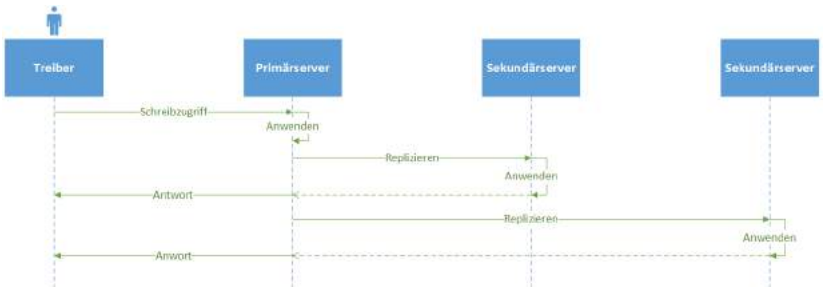

Abbildung 14:

4.2.4 Replication über ein Raspberry Pi-Cluster

Für den Aufbau der Replication kamen drei folgende Server zum Einsatz:

p101 Server	
IP-Adresse	192.168.1.6
Port	27017
Rolle	Primärserver
p201 Server	
IP-Adresse	192.168.1.7
Port	27017
Rolle	Sekundärserver
p301 Server	
IP-Adresse	192.168.1.8
Port	27017
Rolle	Sekundärserver

Tabelle 7: Replication-Umgebung

Als Erstes wird die Konfigurationsdatei mongodb.conf mit der Option –replset rs0 erweitert. Das gilt auch für die Server p201 und p301, da diese Server auch zum Replica Set hinzugefügt werden. Folgender Befehl initialisiert den Server p101 zum Primärserver.

```
rs.initiate()
```

Im nächsten Schritt erfolgt die Erweiterung mit zwei weiteren Teilnehmern für das Replica Set:

```
rs.add("192.168.1.7:27017")
rs.add("192.168.1.8:27017")
```

Der Primärserver p101 führt schreibende Operationen innerhalb einer Replica Set und repliziert sie an die Sekundärservern p201 und p301 (siehe Abbildung 15).

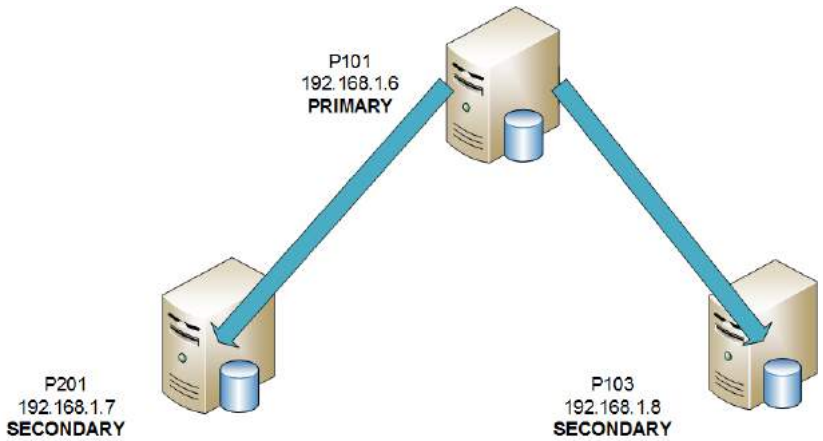

Abbildung 15: Replica Set

Beim Ausfall des Primärservers p101 übernimmt einer der verbleibenden Sekundärservern die Rolle des Primärservers (siehe Abbildung 16).

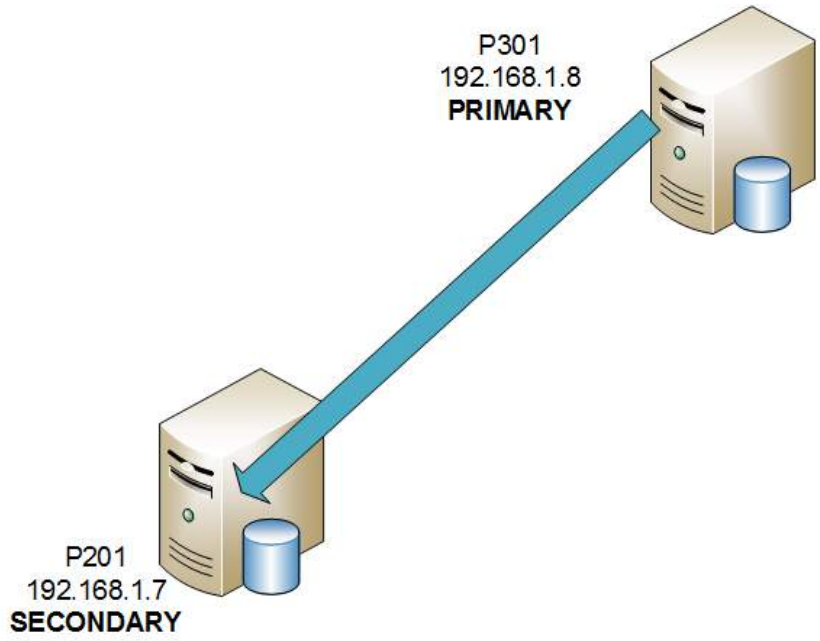

Abbildung 16: Ausfall des Primärservers p101

Bei der Wiederaktivierung des Servers p101 übernimmt dieser Server diesmal die Rolle des Sekundärservers (siehe Abbildung 17).

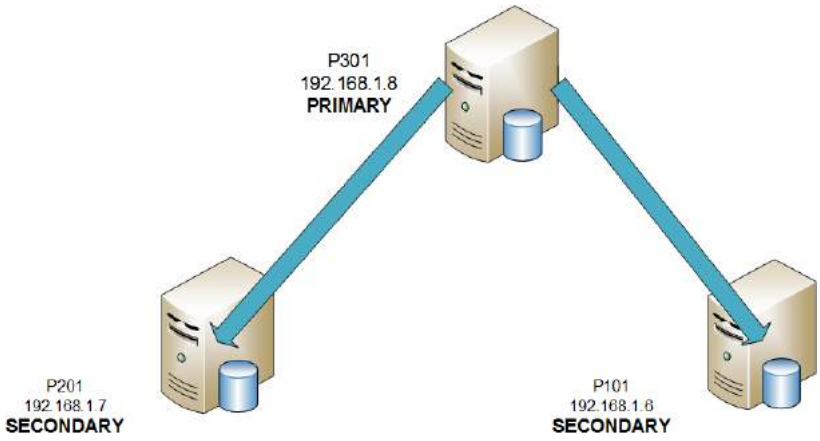

Abbildung 17: Wiederaktiverung des Servers p101

Das Kommando rs.status() zeigt den aktuellen Status des Replica Sets (siehe Abbildung 18).

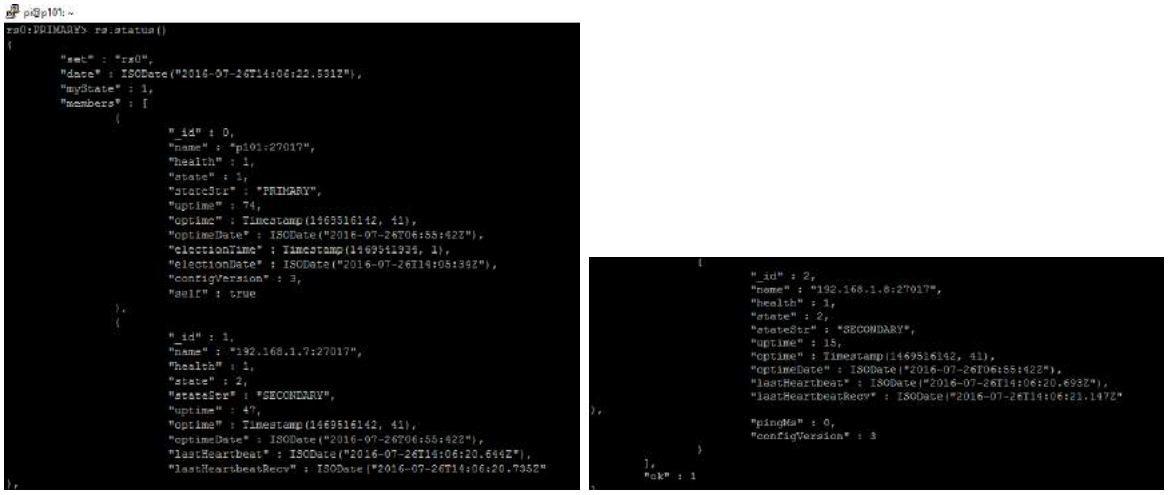

Abbildung 18: Status des Replica Sets

Über das HTTP-Interface unter http://localhost:28017/_replSet besteht die Möglichkeit, den Zustand des Replica Sets einzusehen [6].

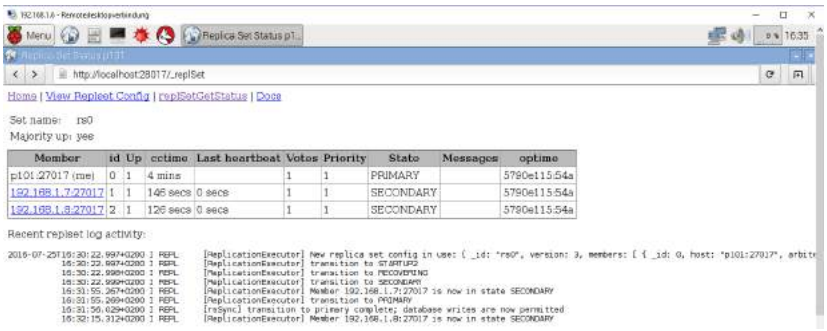

Abbildung 19: Status des Replica Set über HTTP-Interface

Es besteht die Möglichkeit, das Verhalten von Replica Set zu beeinflussen. Das Kommando rs.conf() zeigt neben Host und ID noch folgende Eigenschaften: (siehe Abbildung 20)

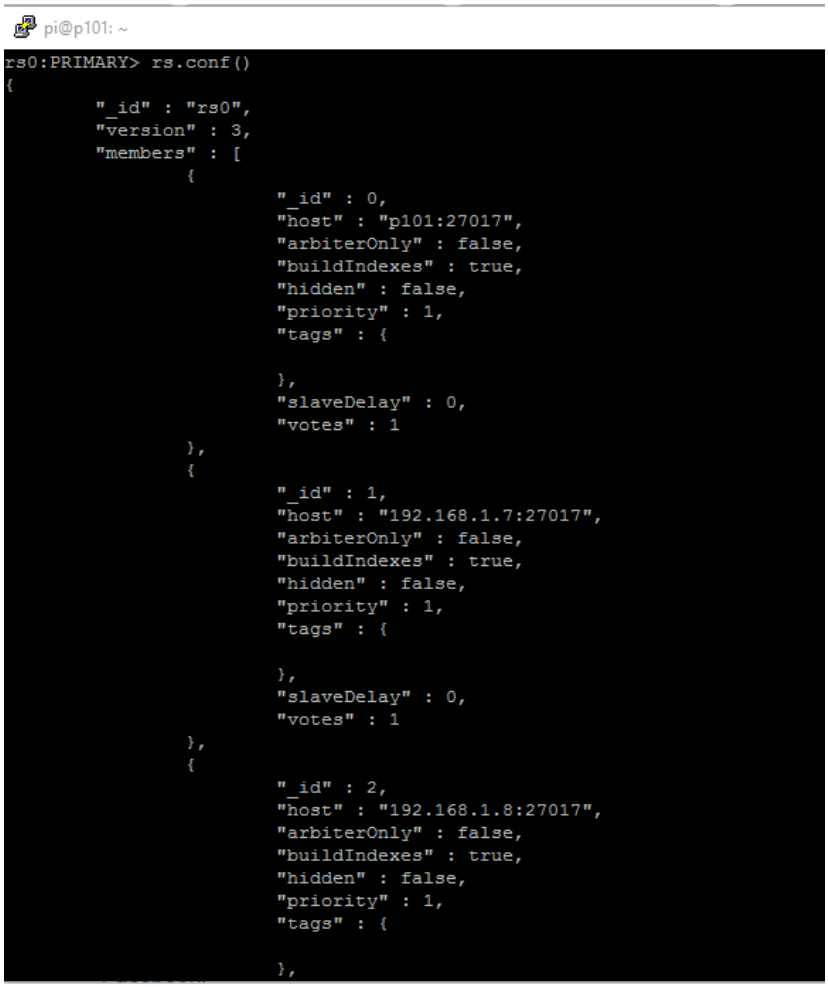

Abbildung 20: JavaScript-Befehl rs.conf()

arbiteronly Ein Arbiter gibt seine Stimme im Falle einer Neuwahl des Primärknotens. Hier ist boolean auf 0 gesetzt und somit nicht aktiv [6].

votes Bestimmt ob es ein Stimmrecht für den Knoten gibt. Die Bestimmung erfolgt durch die Werte 0 und 1. Der Defaultwert ist 1 [6].

slaveDelay Zeigt einen Integerwert, der die Verzögerung in Sekunden an einem Sekundärserver anzeigt [6].

hidden Zeigt einen Booleanwert an, ob ein Knoten versteckt ist [6].

tags Durch das Einsetzen von Tag lassen sich eigene Read Preferences und Write Concers definieren [6].

buildIndexes Der boolean Defaultwert schaltet die Indexerzeugung aus. Diese Operation ist nur sinnvoll bei Back-up-Zwecken, da keine Leseoperationen dafür notwendig sind [6].

Innerhalb dieses Replica Sets sind die Rollen nicht zugeordnet. Im Folgenden erhalten die Server einen Prioritätswert, damit der Server p101 bei einer erneuten Neuwahl wieder die Rolle als Primärserver übernimmt.

```
rs0:PRIMARY> cfg.members[0].priority = 2
2
rs0:PRIMARY> cfg.members[1].priority = 1
1
rs0:PRIMARY> cfg.members[2].priority = 1
1
rs0:PRIMARY> rs.reconfig(cfg)
{ "ok" : 1 }
```

Abbildung 21: Prioritätenvergabe

Je höher der Prioritätswert desto höher, ist die Wahrscheinlichkeit den Primärserver zu übernehmen. Ein Wert von 0 verwehrt die Vergabe des Primärservers vollständig. Der Server p101 erhält somit den höchsten Wert und im Falle einer Wiederherstellung übernimmt dieser Server wieder den Primärserver [6].

Lesezugriff vom Sekundärserver Wie schon bereits erwähnt, verfügt nur der Primärserver die Schreibzugriffe. Jedoch kann ein Defaultwert den Schreibzugriff verhindern und es nur auf die Lesezugriffe beschränken. Der Zugriff auf die Leseoperationen von Sekundärserver erfolgt über das Kommando rs.slaveOk(). Dieses Szenario kommt nur bei Anwendungen infrage, bei denen Strong Consistency nicht erforderlich ist. Dabei handelt sich hier um Eventual Consistency, es deutet auf die Eigenschaft eines verteilten Systems hin. Dies bedeutet, dass alle Mitglieder des Replica Sets schließlich am Ende den gleichen Datenbestand haben aber auch eine inkonsistente Zeitspanne zulassen [6]. Folgendes Beispiel zeigt den Lesezugriff von einem Sekundärserver:

```
rs0:SECONDARY> db.restaurants.count()
2016-08-01T08:21:08.177+0200 E QUERY    Error: count failed: { "note" : "from ex
ecCommand", "ok" : 0, "errmsg" : "not master" }
    at Error (native)
    at DBQuery.count (src/mongo/shell/query.js:326:35)
    at DBCollection.count (src/mongo/shell/collection.js:1046:27)
    at (shell):1:16 at src/mongo/shell/query.js:326
rs0:SECONDARY> rs.slaveOk()
rs0:SECONDARY> db.restaurants.count()
25359
```

Abbildung 22: Lesezugriff durch Sekundärserver

4.3 Vertikale und Horizontale Skalierung

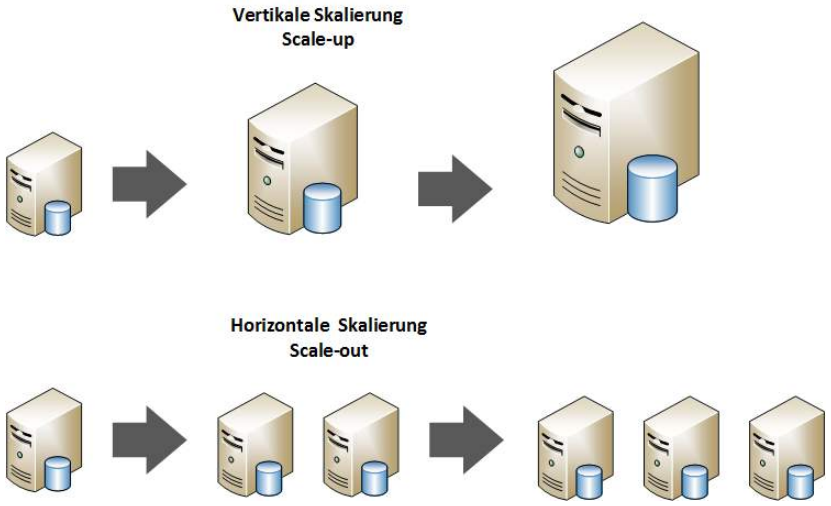

Abbildung 23: Vertikale und Horizontale Skalierung

4.3.1 Vertikale Sklalierung

Die vertikale Skalierung auch Scale-up genannt definiert das Aufrüsten von einem existierenden Server durch bessere Hardwarekomponenten wie zum Beispiel mit mehr CPU, Hauptspeicher und Netzwerkschnittstellen. Ein wesentlicher Faktor ist, dass die Kosten für besseren Hardwarekomponenten nicht gleichmäßig verlaufen und somit auch ein exponentielles Wachstum der Kosten möglich ist. Der Nachteil ist dabei, wenn wenig Server vorhanden sind, auch die Ausfallsicherheit sich dadurch beschränkt [34].

4.3.2 Horizontale Sklalierung

Die horizontale Skalierung auch Scale-out genannt, bedeutet das es eine Anzahl von verfügbaren Servern gibt, die in einer verteilten Datenbanksystem miteinander verbunden sind. Es kann sich dabei um eine sehr hohe Anzahl von Servern handeln. Mehrere tausend Server sind in der heutigen Zeit kein Ausnahmefall. Die Leistung des einzelnen Servers muss nicht zwingend gleichmäßig sein. Der wesentliche Vorteil bei horizontaler Skalierung liegt in der Verfügbarkeit und der Ausfalltoleranz. Die Redundanz kompensiert den Ausfall eines einzelnen Serverknotens oder den Abbruch einer Verbindung zwischen zwei Serverknoten [34].

4.4 Sharding

Sharding ist eine Methode bei MongoDB, um Datenbankpartitionierung durchzuführen. Dadurch lassen sich große Datenmengen in einzelne Knoten verteilen [35]. Die Daten sollen sich anhand von logischen Kriterien verteilen lassen und sollten möglichst in gleicher Datenmenge in den einzelnen Shards vorhanden sein. Bei einem Shard handelt es sich um einen einzelnen Datenbankserver, der eine Teilmenge von Daten beinhaltet. Dabei nennt sich die Teilmenge von Daten Chunk. Somit wird der Wertebereich von den Chunks durch eine obere und untere Grenze eingeschränkt. Es ist zu beachten, dass die maximale Chunk-Größe 64 Megabyte beträgt [36]. Die Aufteilung von mehrere Daten wird von den sogenannten Sharding-Keys bestimmt. Die Reihenfolge der Dokumenten bleibt erhalten. Ein Beispiel hierfür ist die Partitionierung eines Telefonbuches. Der Nachname könnte als Sharding-Key dienen. Bei drei verfügbaren Shards, kann der erste Shard alle Nachnamen von A-H, der zweite alle Nachnamen von I-P und der dritte die restlichen Nachnamen von Q-Z enthalten [35].

4.4.1 Sharding über ein Raspberry Pi-Cluster

Für die Erstellung einer Sharding-Umgebung ist die Reihenfolge relevant. Um die verfügbaren Daten zuzuteilen bzw. zu gliedern sind folgende Schritte notwendig [6]:

- Bekanntgabe der Sharding-Knoten
- Aktivierung einer Datenbank für die Sharding-Umgebung
- Definition des Shard Keys auf der Collection .

Es handelt sich um eine minimale Sharding-Umgebung mit einem Konfigurationsserver, einem Routerprozess und zwei Shard-Servern, die über einen separaten mongod-Prozess laufen.

Folgende Tabelle zeigt die Konfigurationsrollen für die Sharding-Umgebung:

p101 Server	
IP-Adresse	192.168.1.6
Port	27017
Rolle	mongos-Router
p101 Server	
IP-Adresse	192.168.1.6
Port	29017
Rolle	Konfigurationsserver
p201 Server	
IP-Adresse	192.168.1.7
Port	28017
Rolle	Shard-Server
p301 Server	
IP-Adresse	192.168.1.8
Port	28017
Rolle	Shard-Server

Tabelle 8: Sharding-Umgebung

Zunächst erfolgt der Start mit einem Konfigurationsserver. Dieser Prozess läuft auf der IP-Adresse 192.168.1.6 und dem Port 29017. Die Voraussetzung ist ein neu zu erstellender Ordner für die vorgesehene Speicherung von Metainformationen über die Shards und der Verwaltungsinformationen von Chunks [6]. Somit kann der Konfigurationsserver starten:

```
mkdir -p /data/sh/config
sudo mongod –dbpath /data/sh/config –port 29017 –configsvr
```

Durch die Bekanntgabe des Konfigurationsservers kann der Routerprozess (Mongos) starten:

```
sudo mongos –configdb 192.168.1.6:29017 –chunkSize 16
```

Der Parameter chunkSize bestimmt die Größe eines Chunks und enthält einen kleinen Wert von 16 Megabyte. Im nächsten Schritt erfolgt die Konfiguration der zwei Shards. Hierfür wurden die Server p201 und p301 verwendet mit dem jeweiligen Port 28017. Für die Verwaltung der Shards muss man sich mit dem Router über den Mongoshell verbinden:

```
mongo –port 27017
```

Das JavaScript-Objekt *sh* gibt die Möglichkeit, das Sharding zu verwalten. Der Befehl sh.status() zeigt den aktuellen Status der Sharding-Umgebung. Anhand des Feldes partitioned kann man ablesen, dass für die aktuelle Sharding-Umgebung noch keine Datenbank aktiviert ist.

Das Feld shards zeigt alle Server, die sich in der Sharding-Umgebung befinden [6]. Im nächsten Schritt erfolgt die Erweiterung der Sharding-Umgebung mit den Servern p201 und p301 (siehe Abbildung 24 und 25).

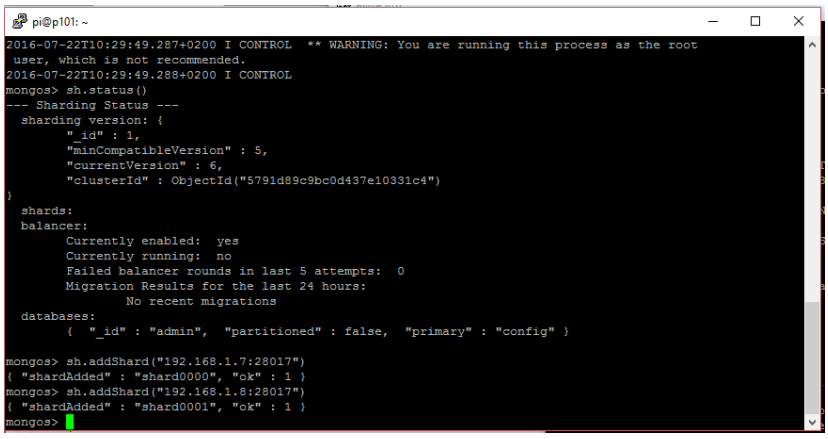

Abbildung 24: Erstellung einer Sharding-Umgebung

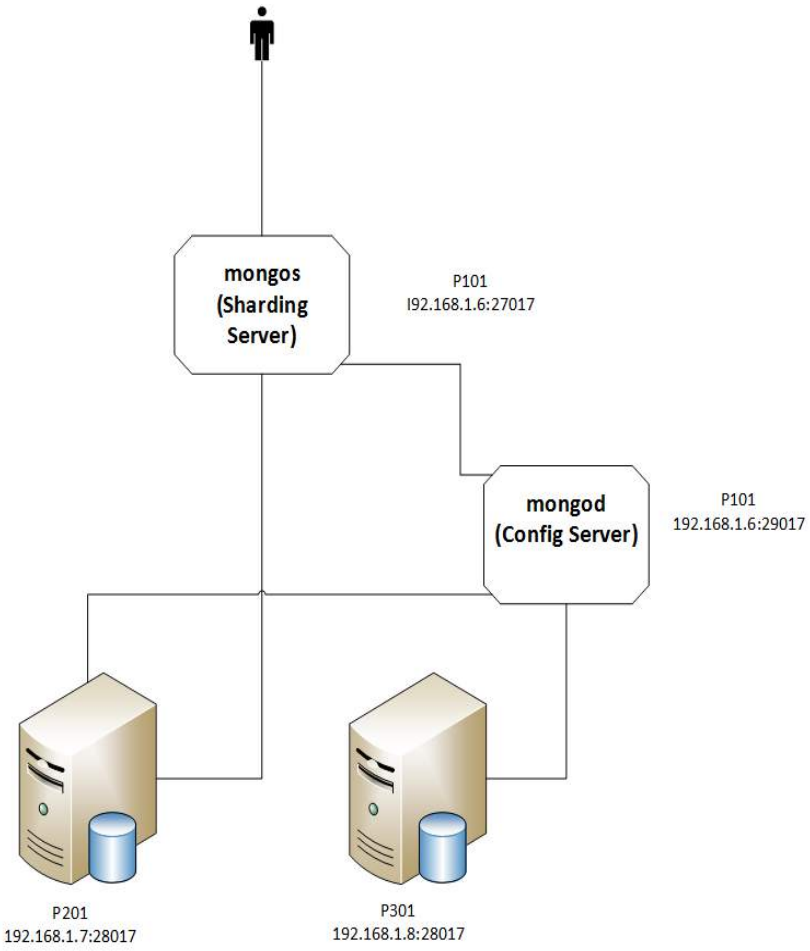

Abbildung 25: Skalierung-Umgebung

Um eine Sharding auf einer Datenbank durchzuführen, benötigt man eine aktive Datenbank. Hierfür wird die Datenbank Twitter verwendet und auf de Server p201 und den Port 28017 importiert. Die Datenbank Twitter ist unter der Webseite **"https://github.com/ttrelle/mongodb-buch/blob/master/data/tweets.zip"** für die Installation verfügbar. Diese Datenbank verfügt 51.428 Dokumente. Nach dem Extrahieren der zip-Datei, wird die BSON-Datei auf den Server p201 importiert. Die Collections werden mit dem Namen *tweets* importiert:

```
mongorestore -port 28017 -d twitter -c tweets /home/pi/tweets.bson
```

Somit ist der Shard shard000 von Server p201 der primäre Shard. Folgender Befehl aktiviert die Datenbank twitter für die Sharding-Umgebung:

```
sh.enableSharding("twitter")
```

Da die Collection aus bestehenden Daten verfügbar ist, muss noch für den Shard ein Index vorhanden sein:

```
db.tweets.ensureIndex("user.screen_name": 1)
sh.shardCollection("twitter.tweets", "user.screen_name")
```

Abbildung 26: Aufteilung der Shards

Es sind vier Chunks vorhanden, die gleichmäßig auf die Shards shard000 und shard0001 verteilt sind. Ebenfalls sieht man für jeden einzelnen Chunk den zugeteilten Wertebereich.

5 Untersuchung der Aggregationsmethoden

Der folgende Abschnitt befasst sich hauptsächlich mit der Untersuchung von Aggregationsmethoden.

5.1 Grafische Benutzeroberfläche für MongoDB

Es gibt eine Reihe von Programmen, die eine grafische Benutzeroberfläche für die Administration von MongoDB anbieten. Für diese Bachelorarbeit kam das Programm Robomongo zum Einsatz (siehe Abbildung 27). Die Verbindung auf eine MongoDB-Instanz erfolgte über das SSH-Protokoll wie beim Programm PuTTY.

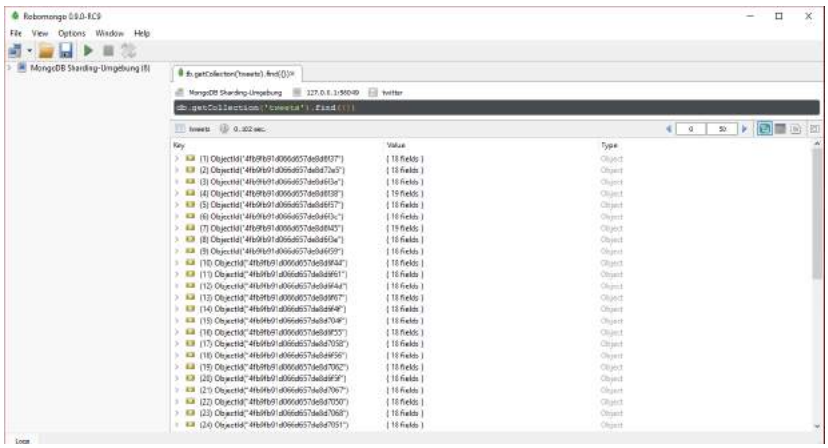

Abbildung 27: Robomongo

5.2 Aggregation von Daten

Bei einer komplexen Abfragemethode kommen Aggregationsmethoden zum Einsatz, als zum Beispiel für Berechnungen von Summen, Durchschnitt, Maximum und Minimum. Das Ergebnis kommt in der gewünschten definierten Ausgabe [6].
MongoDB bietet drei Möglichkeiten [6] für die Aggregation von Daten:

- **Einfache Abfragemethoden zur Datenaggregation**
- **Aggregation Pipline**
- **MapReduce**

5.2.1 Einfache Abfragemethoden zur Datenaggregation

count() Die Aggregationsmethode count() ermittelt die Anzahl der gefundenen Dokumenten [6]. Das folgende Beispiel liefert als Ergebnis eine Anzahl von Benutzer, die mehr als 100 Follower haben [6]:

```
mongos> db.tweets.find({"user.followers_count" : {$lte:100}}).count()
21173
```

Abbildung 28:
fig:count

distinct() Die Aggregationsmethode distinct() reduziert die Abfrage und zeigt nur einen Wert, auch wenn sie mehrfach vorkommen. Dieselbe Funktion existiert auch bei SQL. Der Zugriff kann nur auf die Werte und nicht auf Schlüssel zugreifen [6]. Abbildung 27 zeigt eine Abfrage alle verfügbaren Sprachen der Benutzer durch distinct():

```
mongos> db.tweets.distinct("user.lang")
[ "de", "en", "es", "fr", "it", "ja" ]
```

Abbildung 29: Aggregationsmethode distinct()

group() Durch die Aggregationsmethode group() lassen sich Daten anhand eines Schlüssels gruppieren. Diese Methode führt zusätzlich einfache Operationen wie das Zählen oder Summieren von Werten aus. Der Zugriff erfolgt über die Collection und operiert auf der Basis von Dokumenten [6]. Folgende Tabelle zeigt die vorgegebenen Parameter [6]:

Parameter	**Bedeutung**
key	Schlüsselwert für die Gruppierung.
reduce	Eine Funktion für bestimme Operationen z.B. für das Zählen oder Durchschnittsbildung. Diese Funktion verfügt zwei Eingangsparameter, einmal das aktuelle Dokument und sowie das aggregierte Ergebnisdokument.
initial	Initialisiert das Ergebnis nach einer reduce Funktion.
keyf (Optional)	Berechnet einen neuen Schlüssel, anstatt den bereits definierten Schlüssel zu verwenden.
cond (Optional)	Filterung der Dokumente einer Collection. Sonst sind alle Dokumente gelistet ohne diese Option.
finalize (Optional)	Zeigt das ausgeführte Ergebnisdokument für jeden Datensatz an.

Tabelle 9: Parameter bei group()

Folgende Abbildung zeigt eine Abfrage für die Gruppierung der Sprache und die Anzahl der dazugehörigen Tweets auf der bestehenden Replication-Umgebung (siehe Kapitel 4.2.4). Diese Aggregationsmethode ist nicht kompatibel, um Abfragen auf einer Sharding-Umgebung auszuführen [6].

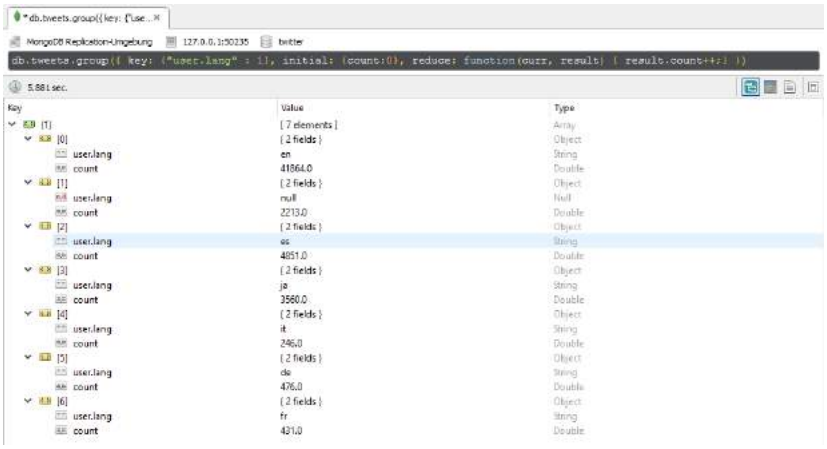

Abbildung 30: Aggregationsmethode group()

5.2.2 Aggregation Pipeline

MongoDB bietet eine eigene komplexe Aggregationsmethode, das sogenannte Aggregation Pipeline als Alternative für MapReduce an. Hier besteht die Möglichkeit, Daten auf mehreren Rechnern parallel zu verarbeiten. Der Begriff Pipeline bedeutet, dass eine Reihe von einzelnen Schritten durch den Anwender definiert ist [13]. Folgende Tabelle zeigt Pipeline-Operationen, die zur Verfügung stehen [37]:

Operationen	**Beschreibung**
$match	Filterung von Dokumenten.
$project	Einschränkung einer Teilmenge von Feldern und somit verändern sich die Werte der Felder.
$sort	Sortierung von Dokumenten.
$skip	Überspringt eine beliebige Anzahl von Dokumenten.
$limit	Begrenzt eine beliebige Anzahl von Dokumenten.
$group	Gruppierung von einem oder mehreren Feldern.
$unwind	Anwendung bei Arrays. Generierung jedes Array-Eintrags für ein neues Dokument der nächsten Pipeline-Stufe.
$redact	Filterung von Felder und Teilbäume des Dokuments in Abhängigkeit vom Inhalte anderer Felder.
$out	Ergebnisanzeige der Aggregation. Außerdem wird auch eine Collection für diese Operation gebildet.

Tabelle 10: Pipeline-Operationen

Folgende Abbildung zeigt eine Abfrage für die Gruppierung der Sprache und die Anzahl der dazugehörigen Tweets auf der bestehenden Sharding-Umgebung(siehe Kapitel 4.4.1):

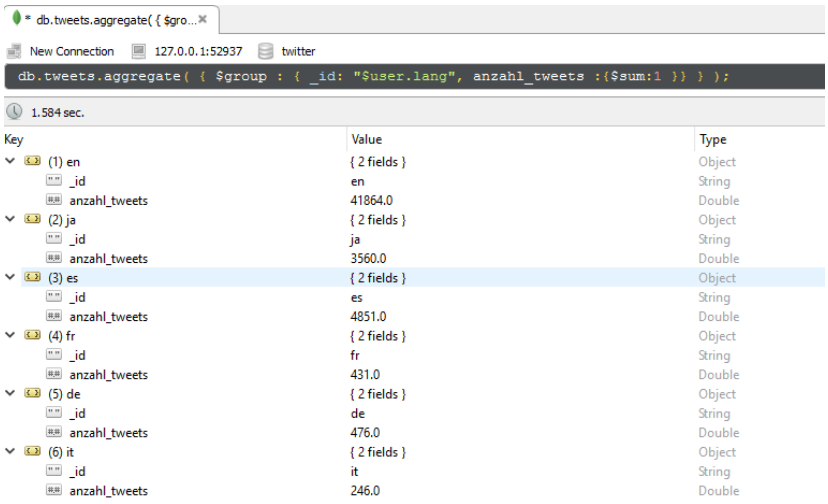

Abbildung 31: Aggregation Pipeline

Die Messung für die Abfrage ergab 1584 Millisekunden. und zeigt das die Englische Sprache die meistbenutzte Sprache bei Twitter ist.

5.2.3 MapReduce

Die Aggregationsmethode Aggregation Pipeline ist sehr umfangreich aber es gibt auch Szenarien in denen es nicht ausreichen würde. Eine weitere Funktion für die Aggregation ist das MapReduce-Verfahren. Ein wesentlicher Vorteil dabei ist, dass das MapReduce-Verfahren alle Datensätze auch in einem verteilten Datenbanksystem durchsuchen [37]. Folgende Abbildung zeigt eine Abfrage mit dem gleichen Ergebnisziel wie in der Abbildung 31.

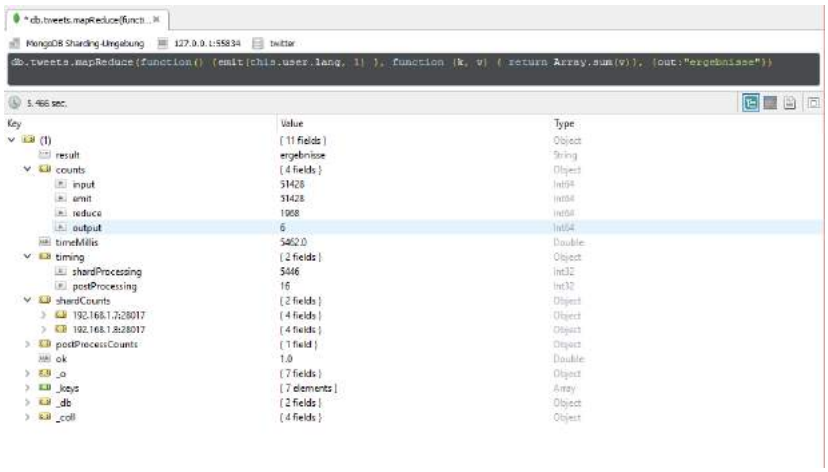

Abbildung 32: MapReduce

Nach der Ausführung liefert die Ausgabe mehrere Statistiken. Die Berechnung zeigt, das MapReduce-Verfahren wesentlich länger braucht für die Abfrage als Aggregation Pipeline. Die Berechnung ergab 5148 Millisekunden. Zudem wurden 51.428 Dokumente gelesen (input) und genau so viele Schlüssel-Wert-Paare (emit) produziert. Die reduce-Funktion wurde 1.968 mal aufgerufen. Die Funktion **out** erstellt eine Collection-Ausgabe namens *ergebnisse*, aus der die Ergebnisse auszulesen sind. Die Folgende Abbildung zeigt die Collection *ergebnisse*:

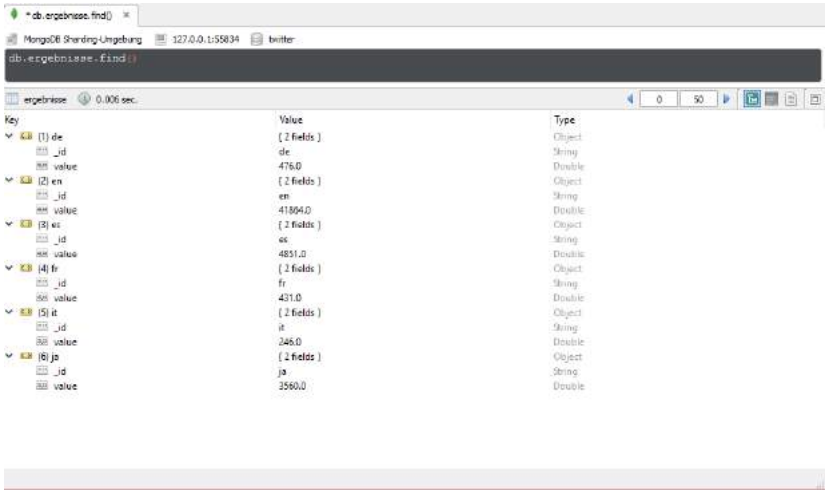

Abbildung 33: Ergebnisanzeige der Aggregation

6 Fazit

Die Zielsetzung der vorliegenden Arbeit war der Aufbau eines NoSQL Datenbanksystem-Clusters mit MongoDB auf Basis von Einplatinencomputern sowie die Untersuchung deren Funktionalität. Wie die Bachelorarbeit gezeigt hat, eignet sich der Einplatinencomputer Raspberry Pi hervorragend für das Experimentieren mit eigenen Clustersystemen, da die Anschaffungskosten minimal sind und es einen geringen Stromverbrauch erzeugt. Die Arbeit mit dem Einplatinencomputer Raspberry Pi und dem NoSQL-Datenbanksystem MongoDB hat mir einen großen Spaß gemacht.

Mit MongoDB lernte und analysierte ich die Funktionalitäten eines verteilten Datenbanksystems durch die Replikation der Daten (Replication) und ihre Skalierbarkeit (Sharding). Neue Server lassen sich einfach und schnell in einem vorhandenen Cluster einbinden. Der Nachteil ist, dass kaum Programme für Messungen existieren, welche die Leistung von NoSQL Datenbanken vergleichbar machen können. Einer diesen wenigen Programme ist Yahoos „Cloud Serving Benchmark“ [YCSB]. Aber auch hier kann man sich nicht vollständig auf den Leistungsvergleich verlassen.

Die Zukunft sehe ich für MongoDB optimistisch, da immer mehr Großunternehmen das Datenbanksystem benutzen und allgemein die Nachfrage nach NoSQL-Datenbanksystemen sich steigt. Da die Anforderungen an traditionelle Datenbanksysteme nicht mehr bewältigt werden können. Besonders im Bereich der Webentwicklung eignen sich unterschiedliche Datenstrukturen.

Wie sich andere NoSQL-Datenbanksysteme nach Funktionalität und Leistung unterscheiden, bedarf es an weiteren Untersuchungen. Mein Interesse hat sich auf Big Data fokussiert und in diesem Zusammenhang, kam ich auf die NoSQL-Datenbanksysteme.

Literaturverzeichnis

[1] Edgar f.codd. https://de.wikipedia.org/wiki/Edgar_F._Codd. Eingesehen am 18.07.2016.

[2] Sql. https://de.wikipedia.org/wiki/SQL. Eingesehen am 18.07.2016.

[3] Ddl. de.wikipedia.org/wiki/Data_Definition_Language. Eingesehen am 18.07.2016.

[4] What ist big data? https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html. Eingesehen am 14.07.2016.

[5] Raspberry pi guide. https://raspberrypiguide.de. Eingesehen am 30.06.2016.

[6] Tobias Trelle. *MongoDB - Der praktische Einstieg*. dpunkt.verlag, Heidelberg, 1. auflage edition, 2014.

[7] Db-engines ranking. https:///db-engines.com/de/ranking. Eingesehen am 01.07.2016.

[8] Mongodb. http:wikis.gm.fh-koeln.de/wiki_db/Datenbanken/MongoDB. Eingesehen am 13.07.2016.

[9] Who uses mongodb. http://www.mongodb.com/who-uses-mongodb. Eingesehen am 13.07.2016.

[10] Mongodb mit dem „größten release“ seiner geschichte. http:/jaxenter.de/mongodb-mit-dem-grosten-release-seiner-geschichte-1296. Eingesehen am 01.07.2016.

[11] Was ist bei big data? http://www.computerwoche.de/a/was-ist-was-bei-big-data,3070624/. Eingesehen am 20.06.2016.

[12] Stefan Edlich, Achim Friedland, Jens Hampe, Benjamin Brauer, and Markus Brückner. *NoSQL - Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*. Hanser, Muenchen, 2. aufl. edition, 2011.

[13] Johannes Schildgen. *MongoDB kompakt - Was Sie über die NoSQL-Dokumentendatenbank wissen müssen*. BoD – Books on Demand, Norderstedt, 1. aufl. edition, 2016.

[14] Base. https:///wikis.gm.fh-koeln.de/wiki_db/Datenbanken/BASE. Eingesehen am 07.07.2016.

[15] Keyvaluesysteme. https:///wikis.gm.fh-koeln.de/wiki_db/Datenbanken/KeyValueSysteme. Eingesehen am 07.08.2016.

[16] Nosql. https://de.wikipedia.org/wiki/NoSQL. Eingesehen am 25.06.2016.

[17] Mongodb. http://pi.informatik.uni-siegen.de/Mitarbeiter/mrindt/Lehre/Seminare/NoSQL/einreichungen/NOSQLTEC-2015_paper_13.pdf. Eingesehen am 27.07.2016.

[18] Raspberry pi installation. http://www.netzmafia.de/skripten/hardware/RasPi/RasPi_install.html. Eingesehen am 22.06.2016.

[19] Raspberry pi downloads. http:///www.raspberrypi.org/downloads. Eingesehen am 22.06.2016.

[20] Matt Richardson and Shawn Wallace. *Raspberry Pi für Einsteiger*. O'Reilly Germany, Koeln, 2. aufl. edition, 2015.

[21] Statische/feste ipv4-adresse für raspbian jessie. https://www.elektronik-kompendium.de/sites/raspberry-pi/1912151.htm. Eingesehen am 20.07.2016.

[22] Raspberry pi - remote desktop (xrdp) installieren. www.datenreisw.de/raspberry-pi-remote-desktop-xrdp-installieren. Eingesehen am 11.07.2016.

[23] Mongo. https://docs.mongodb.com/manual/reference/program/mongo. Eingesehen am 23.07.2016.

[24] Mongod. https://docs.mongodb.com/manual/reference/program/mongod. Eingesehen am 23.07.2016.

[25] Mongoperf. https://docs.mongodb.com/manual/reference/program/mongoperf. Eingesehen am 23.07.2016.

[26] Mongos. https://docs.mongodb.com/manual/reference/program/mongos. Eingesehen am 23.07.2016.

[27] Mongoexport. http://www.mongodb.com/who-uses-mongodb. Eingesehen am 01.07.2016.

[28] Mongorestore. https://docs.mongodb.com/manual/reference/program/mongorestore. Eingesehen am 24.07.2016.

[29] Mongodump. https://docs.mongodb.com/manual/reference/program/mongodump. Eingesehen am 24.07.2016.

[30] mongofiles. http://docs.mongodb.com/manual/reference/program/mongofiles. Eingesehen am 13.07.2016.

[31] Mongooplog. https://docs.mongodb.com/manual/reference/program/replica-set-oplog. Eingesehen am 24.07.2016.

[32] Mongodb 3.0.9 binaries for raspberry pi 2 jessie. https://andyfelong/2016/01/mongodb-3-0-9-binaries-for-raspberry-pi-2-jessie. Eingesehen am 26.06.2016.

[33] Configuration options. https:/docs.mongodb.com/manual/reference/configuration-options. Eingesehen am 25.06.2016.

[34] Skalierbarkeit. http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/Skalierbarkeit. Eingesehen am 06.08.2016.

[35] Sharding. http://db-engines.com/de/article/Sharding. Eingesehen am 12.07.2016.

[36] Rahm Rahm, Saake Saake, and Sattler Sattler. *Verteiltes und Paralleles Datenmanagement - Von verteilten Datenbanken zu Big Data und Cloud*. Springer-Verlag, Berlin Heidelberg New York, 1. aufl. edition, 2015.

[37] Mongodb fuer softwareentwickler. http://www.informatik-aktuell.de/betrieb/datenbanken/mongodb-fuer-software-entwickler.html. Eingesehen am 07.08.2016.