Fachhochschule Aachen

Campus Jülich

Fachbereich 9

Medizintechnik und Technomathematik

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Seminararbeit

Im Studiengang Angewandte Mathematik und Informatik

Vergleich von relationalen Datenbanksystemen und Time-Series-Datenbanksystemen zur Ablage von Messdaten

Autor:	Moritz Koch
Matrikelnummer:	3193079
Erste Prüfer:	Prof. Dr. rer. Nat. Alexander Voß
Zweiter Prüfer:	Dipl.-Inform. Uwe Steinhausen

Aachen, den 14. Dezember 2020

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Seminararbeit mit dem Thema

Vergleich von relationalen Datenbanksystemen und Time-Series-

Datenbanksystemen zur Ablage von Messdaten

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Ich verpflichte mich, ein Exemplar der Seminararbeit fünf Jahre aufzubewahren und auf Verlangen dem Prüfungsamt des Fachbereiches Medizintechnik und Technomathematik auszuhändigen.

Name: Moritz Koch

Aachen, den 14.12.2020

Unterschrift der Studentin / des Studenten

Inhaltsverzeichnis

Abstract

Diese Seminararbeit soll ein relationales Datenbanksystem und ein Time-Series-Datenbanksystem vor dem Hintergrund der Ablage von Messdaten vergleichen. Dies geschieht unter der Prämisse, dass National Instruments einem Kunden geholfen hat, eine Lösung zur Auswertung von Daten zu erstellen, welche von Batterieprüfständen generiert werden. Diese Daten werden zurzeit in Dateien abgelegt, bevor sie zur Auswertung weitergereicht werden.

Messdaten beschreiben Daten, welche bei Messungen anfallen und zeichnen sich durch einen Messwert, eine Einheit und häufig eine Quelle aus. Zeitreihendaten erweitern diese Definition um einen Zeitstempel, welcher den Messzeitpunkt angibt.

Um eine datenbankbasierte Lösung zu erarbeiten wurden das Datenbanksystem MySQL als relationales und das Datenbanksystem InfluxDB als Time-Series-Datenbanksystem ausgewählt.

Da die in den Prüfständen gemessenen Daten sich nicht zur Verwendung in dieser Seminararbeit eignen, da sie für den Untersuchungszeitraum nicht schnell genug anfallen, wurden Beispieldaten generiert, mit welchen im Folgenden gearbeitet wird. Um diese Daten zu messen sind automatisierte Skripte in Python verfasst worden, welche mit Hilfe des Moduls psutil Systemprozessdaten messen und über die Python-Konnektoren der beiden Datenbankclients die Messwerte in die jeweilige Datenbank schreiben.

Die Datenbanken sind als relationale und Time-Series-Datenbank anders strukturiert und verwenden auch andere Anfragesprachen zur Filterung der Daten. MySQL nutzt SQL und InfluxDB die hauseigene Sprache Flux. Beide Sprachen ermöglichen Abfragen, Arithmetik und Filtern auf den Daten.

Auch die unterliegenden Speicherkonzepte unterscheiden sich, beispielsweise im Hinblick auf Datenreduzierung durch Downsampling. InfluxDB bietet die Möglichkeit mit Aufbewahrungsrichtlinien, sogenannten „Retention Policies“, nicht mehr benötigte Daten automatisiert zu löschen und im Zusammenspiel mit beständigen Queries Downsampling einfach umzusetzen. MySQL kann auch beständige Queries benutzen und Retention Policies könnten mit diesen Queries nachgebaut werden, werden aber nicht nativ unterstützt.

Im Vergleich der beiden Datenbanksysteme zum Verwalten von Messdaten bietet InfluxDB das überzeugendere Feature Set und zusätzlich eine niedrige Einstiegsbarriere für neue Nutzer. Mit MySQL können fehlende Features nachgebaut werden, welche aber nicht an die native Integration der InfluxDB heranreichen. Allerdings wird bei MySQL SQL als Grundlage und als Anfragensprache verwendet und bietet eine höhere Kompatibilität mit anderen Datenbanksystemen, sollte die erarbeitete Problemlösung auf ein anderes System umziehen müssen. Hier ist InfluxDB durch die Verwendung von Flux eingeschränkt.

Letztendlich aber überwiegen die Vorteile der InfluxDB im Hinblick auf die reine Verwaltung und Ablage von Messdaten in einem Datenbanksystem.

1 Motivation

National Instruments strebt danach, das führende Unternehmen im Bereich automatisierter Tests zu werden. Um dieses Ziel zu erreichen arbeitet National Instruments eng mit seinen Kunden zusammen und steht bei Problemen und Fragen zur Seite.

Der Motivation dieser Seminararbeit liegt ein solcher Kundenkontakt zu Grunde: Einer unserer Kunden stellt Elektrogeräte her. Um die Sicherheit und volle Funktionsfähigkeit von Batterien anderer Hersteller in den eigenen Gräten gewährleisten zu können, werden vom Kunden Prüfstände betrieben.

Um diese Prüfstände zu überwachen und Ergebnisse zu sammeln, haben unsere Mitarbeiter in Zusammenarbeit mit dem Kunden einen Monitoring-Prozess etabliert. In diesem Prozess werden die gemessenen Daten, welche im Verlauf der Tests anfallen, in Dateien geschrieben. Aus diesen Dateien sollten die Werte nun wieder ausgelesen werden und zur weiteren Auswertung weitergereicht werden. Da auf die Dateien also nun gleichzeitig geschrieben und davon gelesen werden soll, wurde ein eigenes DataPlugin geschrieben, welches dies ermöglicht. Die Daten können nun weiter ausgewertet werden.

Der oben beschriebene Prozess der Datenablage in Dateien ist teils ineffizient, was im Folgenden näher erläutert wird:

Der beschriebene Arbeitsfluss basiert auf kontinuierlichen Schreib- und Lesezugriffen auf Dateien. Dies kann als problematisch angesehen werden, da aus diesen Zugriffen eine hohe Prozessbelastung folgt.

Außerdem gibt es keinen Ansatz für eine Datenreduzierung für längere Untersuchungen und Beobachtungen. Dadurch kann es zu einer hohen Speichernutzung kommen.

Das Ziel dieser Seminararbeit ist es ein einfaches, zeitgemäßes datenbankbasiertes Ablagekonzept zu entwickeln. Es wird nicht das hier Erarbeitete mit der bestehenden Lösung verglichen, vielmehr wird in dieser Arbeit angestrebt, zwei mögliche Ansätze, Datenbanksysteme, miteinander zu vergleichen. Zu diesem Zweck wurde in der Seminararbeit MySQL, ein relationales Datenbanksystem, und InfluxDB, ein für Zeitreihendaten optimiertes Time-Series-Datenbanksystem, untersucht. Diese sind in ihrem respektiven Bereich sehr beliebte Systeme und deshalb gut für den Vergleich geeignet [25, 26].

Der Autor wird zunächst den Begriff Messdaten näher erläutern, auf die jeweiligen Datenbanksysteme selbst kurz eingehen und ihre Struktur beschreiben. Es werden die Ansätze der Datendurchsuchung mit Queries beschrieben, sowie Konzepte der Speicherung und auch der

Datenreduzierung erläutert. Unterschiede der beiden Systeme und auch der Lösungsansätzen im Ganzen werden aufgezeigt, bevor zum Schluss der Seminararbeit ein Fazit gezogen wird. Außerdem wird ein kurzer Ausblick auf weitere Untersuchungen gegeben.

2 Messdaten

Unter Messdaten versteht man einfach betrachtet, nur Daten oder Datenpunkte, welche bei Messungen anfallen.

Messdaten sind also durch die Messung der sie entspringen definiert. Sie besitzen ein Kennzeichnungsmerkmal, welches sie zu der ursprünglichen Messung zuordnet, außerdem eine Einheit, welche die Größe des Messwertes definiert und den eigentlichen Wert des Messpunktes. Ein Zeitstempel, der den Messzeitpunkt angibt, ist nicht zwingend gegeben [22]. Als Beispiel lässt sich hier ein Messwert aus dem Ausgangsszenario heranziehen. Spannung wird in den Prüfständen unseres Kunden nicht über eine bestimmte Zeit gemessen, sondern pro Ladezyklus. Messdaten, welche einen solchen Zeitstempel besitzen, werden auch Zeitreihendaten genannt.

In unserem Anwendungsfall testet der Kunde Batterien auf gute Integration in eigenen Elektrogeräten. Es fallen also typisch elektrische Maße an. Hierzu zählt die Energie und Spannung pro Ladezyklus, genauso wie der Wechselstromwiderstand über einen Ladezyklus. Außerdem sollen auch die Stromstärke und die Temperatur der Batterie über die Zeit gemessen werden.

Diese Messwerte werden von unserem Kunden über lange Zeit erhoben, zu lange für den vergleichsweise kurzen Untersuchungszeitraum dieser Seminararbeit. Aus diesem Grund werden für die Untersuchung der beiden Datenbanken Beispielmesswerte generiert, welche in höherer Frequenz gemessen werden, um Untersuchungen zu unternehmen.

3 Datenbanken

In diesem Kapitel werden das MySQL-Datenbanksystem und das InfluxDB-Datenbanksystem etwas näher beschrieben und erste Unterschiede aufgezeigt. Außerdem wird die verwendete Datenbankstruktur beschrieben, mit der im weiteren Verlauf der Seminararbeit gearbeitet wird.

3.1 MySQL

MySQL ist ein Open-Source Datenbanksystem von Oracle. MySQL ist ein sogenanntes relationales Datenbanksystem. In einem relationalen Datenbanksystem wird ein tabellenorientiertes Datenmodell verwendet. Diese Tabellen sind durch eindeutige Namen identifizierbar und besitzen eine feste Anzahl an Attributen, oder auch Spalten, mit entsprechenden Datentypen. So eine Tabelle wird auch als Relation und ihre Struktur als (Relations-) Schema bezeichnet. Ein Datensatz, oder auch Entity genannt, ist dann ein Eintrag in einer solchen Relation und wird durch die Ausprägung der einzelnen Attribute der Relation definiert. In dieser Seminararbeit wurde mit der neusten MySQL Distribution (8.0) gearbeitet. Die Datenbank wurde über das mitgelieferte Tool „MySQL Workbench“ konfiguriert. [27]

3.2 InfluxDB

InfluxDB, ein Open-Source Datenbanksystem von InfluxData, verfolgt einen anderen Ansatz als MySQL. InfluxDB ist ein Time-Series-Datenbanksystem. Ein solches Datenbanksystem ist optimiert auf Arbeit mit Zeitreihendaten. InfluxDB ist also darauf ausgelegt, mit hochfrequenten Zeitreihendaten effizient umzugehen, sprich sie zu sammeln, zu speichern, darauf zu filtern und Berechnungen anzustellen. [1, 28]

InfluxDB ist eine NoSQL-Datenbank. Das Schema einer Relation wird also nicht vordefiniert, vielmehr, es gibt keine Relation. Daten werden in einem sogenannten „Bucket“ gespeichert. Jeder Bucket gehört einer Organisation an und besitzt eine Aufbewahrungsrichtlinie, eine sogenannte „Retention Policy“, welche besagt, wie lange Daten gespeichert werden. Dies wird später näher erläutert. In dieser Seminararbeit wurde mit der neusten InfluxDB Distribution gearbeitet (2.0). Diese Version verändert das Verhalten der InfluxDB und bietet neue Funktionen, welche auch in dieser Arbeit verwendet werden. InfluxDB läuft unter Windows nicht direkt auf dem Computer, sondern in einem Docker-Container, in welchem die Datenbank bereits vorkonfiguriert ist. Auf der Datenbank kann über eine Konsolenapplikation, eine Http-API oder über ein Webinterface gearbeitet werden. Das Erstellen einer Datenbank und die Untersuchung der Daten wurden in der Seminararbeit über das Webinterface gemacht. Das Generieren der Daten und das Einpflegen in die Datenbank hingegen über ein automatisiertes Skript.

Da es sich bei der InfluxDB und MySQL um zwei grundlegend unterschiedliche Datenbanksysteme handelt, wird in diesem Kapitel die jeweils verwendete Datenbankstruktur näher erläutern.

3.3 Struktur

Die MySQL Datenbank, die für diese Seminararbeit aufgesetzt wurde, besteht aus drei Relationen. Sie befindet sich in der dritten Normalform, das heißt es werden keine redundanten Daten gespeichert.

Die Relation „data“ beinhaltet die gemessenen Werte. Als Attribute besitzt sie eine ID vom Datentyp INT, die ID der Messwertursprünge als Fremdschlüssel des Typen INT, den Messwert der Quelle als FLOAT, die Einheit des Messwertes als Fremdschlüssel des Typen INT und einen Zeitstempel vom Typ DATETIME. Ein Entity in dieser Relation füllt also all diese Attribute (Spalten). Die ID in dieser Relation ist ein Auto-Inkrement, sie wird also automatisch für jede eingefügte Entity erhöht. Die ID muss auch nicht beim Einfügen eines Datensatzes in die Relation angegeben werden. In der Relation „channels“ sind die Messwertursprünge hinterlegt. Als Attribute umfasst sie eine eindeutige ID als INT und den Typen des Channels als VARCHAR. Die letzte Relation, „units“, umfasst eine eindeutige ID (INT) für die Einheiten und die Typbezeichnung der Einheit als VARCHAR. Die IDs in den Relationen „channels“ und „units“ werden als Fremdschlüssel in „data“ verwendet.

Das InfluxDB-Datenbanksystem ist simpler aufgebaut als das der MySQL. InfluxDB ist keine relationale Datenbank. In der InfluxDB gibt es also keine Relationen. An deren Stelle tritt ein einziges Konstrukt, „Bucket“ genannt. Dieser Bucket hat kein festes Schema. Für neue Attribute kann der Bucket erweitert werden und wenn Attribute nicht bedient werden, können sie automatisch mit Default Werten befüllt werden. Einträge können zum einfacheren Filtern mit Tags und Labels versehen werden. Jeder Eintrag besitzt einen Zeitstempel. Dieser sollte nicht von den Messdaten übergeben werden, sondern es sollte der von InfluxDB beim Einfügen zugewiesene Zeitstempel genutzt werden. Eine Indizierung erfolgt aber nicht über eine eindeutige ID, sondern über den Zusammenschluss aller Attribute. Sollten sich dennoch Einträge doppeln muss anders verfahren werden. Darauf wird im Kapitel „Speicherung“ noch einmal näher eingegangen.

4 Messungen

In diesem Kapitel wird der Arbeitsfluss beschrieben, welcher genutzt wurde, um die in der Seminararbeit verwendeten Beispielwerte zu generieren. Es wird die Verbindung zu dem jeweiligen Datenbanksystem und das Einpflegen der Daten beschrieben.

4.1 Generierung der Daten

Da die im Ausgangszenario verwendeten Messvorgänge und die daraus resultierenden Messwerte für den Zeitraum der Seminararbeit ungeeignet sind, mussten neue Daten generiert werden. Hierzu wurden grundlegende Systemprozesswerte herangezogen, da diese einfach und kostengünstig zu erhalten sind.

Es wurden die CPU-Auslastung in Prozent, die Gesamtheit des genutzten Arbeitsspeichers und die Gesamtheit der gesendeten Bits über das verbundene Netzwerk verwendet. Um diese Werte zu messen wurde jeweils ein Python-Skript geschrieben. Python eignet sich in diesem Fall gut für das Messen der Daten, da es eine große Bibliothek an Modulen für verschiede Anwendungszwecke, wie beispielsweise auch Diesen, zur Verfügung stellt und zu dem eine simple und leicht zu verwendende Sprache ist.

Im Folgenden wird der Aufbau eines der Skripte beschrieben. Alle drei Skripte sind identisch strukturiert und unterscheiden sich lediglich in den gemessenen Daten.

Zunächst müssen die benötigten Module importiert werden. Zum Messen der benötigten Daten wurde das Modul „psutil“ genutzt. Psutil ist eine kostenfreie Cross-Platform-Bibliothek zum beschaffen von laufenden Prozessdaten und Systemauslastungen unter Windows und weiteren Betriebssystemen. Es können Daten unter anderem von CPU, RAM, Speicher, Netzwerk und Sensoren gemessen werden und das Modul wird häufig für Systemmonitoring und ähnliche Anwendungen genutzt. Damit eignet es sich auch gut für unseren Anwendungsfall [24].

Es müssen auch die Module zur Datenbankverbindung importiert werden. Das sind „mysql.connector“, „influxdb_client“ und „influxdb_client.client.write_api“ respektiv. Außerdem wird noch das Modul „datetime “ benötigt um den Messwerten beim Einpflegen in das MySQL-Datenbanksystem einen Zeitstempel mitzugeben.

Das Modul „mysql“ ist die Clientbibliothek des MySQL-Konnektors, welcher über mysql.connector bereitgestellt wird. Zur Verbindung des Clients, sprich unseres Skripts, muss in der connect-Methode des Konnektors der Host, User und Passwort des benutzenden Users, sowie der Name der Datenbank, mit welcher eine Verbindung etabliert werden soll, angegeben werden. Damit ist eine Verbindung hergestellt. Um nun auch später auf die Datenbank schreiben zu können, muss ein

Cursorobjekt deklariert werden. Dieses Objekt kann Operation wie SQL-Statements auf der Datenbank über die das Clientobjekt ausführen. Da uns das Schema der MySQL-Datenbank bereits bekannt ist, können wir auch schon das SQL-Statement vorbereiten, in welches später nur noch die passenden Werte eingefügt werden. [19, 20]

```
mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="asdfghjkl",
    database="seminararbeit"
)

mycursor = mydb.cursor()

sql = "INSERT INTO data (channel, value, unit, time) VALUES (%s, %s, %s, %s)"
```

Abbildung 1: Client-Objekt, Datenbankcursor und SQL-Statement zur Verbindung mit MySQL aus Python

Über das Modul „influxdb_client“ wird die Verbindung zur InfluxDB-Instanz etabliert. Für die Verbindung unseres Clients müssen dem Konstruktor unserer Clientklasse die Host-URL, ein Security-Token und die Organisation des Users übergeben werden. Der Security Token muss zuerst generiert werden. Dies kann beispielsweise über das Webinterface geschehen. Nun ist eine Verbindung zur gesamten InfluxDB-Instanz mit all ihren Buckets geöffnet. Um später in einen Bucket Daten schreiben zu können muss noch ein API-Objekt über das Client-Objekt erstellt werden [29].

```
bucket = "seminararbeit"
org = "MATSE"
token = "2B6PmboaeCRAs0-7QyfNTypWn44__i0V6PNr9_rzkRVUO-UnuKwlOXRJGRNDLNtvDxLNFL6ZbuX-vuW9HM7eDg=="
# Store the URL of your InfluxDB instance
url = "http://localhost:8086"

client = influxdb_client.InfluxDBClient(
    url=url,
    token=token,
    org=org
)

write_api = client.write_api(write_options=SYNCHRONOUS)
```

Abbildung 2: Client-Objekt für InfluxDB und API-Objekt zum Schreiben auf Datenbank

Mit allen benötigten Verbindungen geöffnet, können nun die Messwerte generiert werden. Dies geschieht in einer Endlosschleife ohne Abbruchbedingung. Die Messung kann nur manuell durch das

Stoppen des Skripts beendet werden. Zum Messen wird das bereits oben beschriebene Modul „psutil“ genutzt. Die zu messenden Werte, CPU-Auslastung, die Gesamtheit des genutzten Arbeitsspeichers und die Gesamtheit der gesendeten Bits über das verbundene Netzwerk, können über einfache Funktionsaufrufe abgeholt werden. Dies geschieht für unterschiedliche Werte in unterschiedlichen Frequenzen um die Ausgangssituation, welche im Kapitel Motivation beschrieben wurde, mit zu simulieren. Folgende Frequenzen wurden gewählt: Die CPU-Auslastung wird jeden Clock-Tick abgefragt, da dies der sich am meisten verändernde Wert ist. Die Arbeitsspeichernutzung wird alle 2 Sekunden und die gesendeten Bits über das Netzwerk werden mit der niedrigsten Frequenz abgefragt, alle 5 Sekunden.

4.2 Einpflegen der Daten

Die jetzt gemessenen Werte müssen nun über die bereits etablierten Verbindungen in die Datenbanken eingepflegt werden. Dies geschieht für beide Datenbanken leicht unterschiedlich.

Zum Einpflegen der Messwerte in die MySQL haben wir bereits ein Cursor-Objekt erstellt, welches den bereits vorbereiteten SQL-Befehl ausführen kann. Die vier geforderten Attribute werden zunächst in einem Vektor gespeichert. Dies umfasst die ID der gemessenen Komponente (Channel), den eigentlichen Messwert, die ID der Einheit des Messwertes und den Zeitstempel. Eine ID für den Datensatz muss wie oben beschrieben nicht angegeben werden. Über den Execute-Befehlt des Cursor-Objekts werden die Werte im SQL-Statement integriert und das Ausführen vorbereitet. Mit dem Aufruf der Commit-Funktion wird der Datensatz in die Datenbank eingefügt. Hier wäre auch eine Lösung mit Transaktionen denkbar, welche Datensatzbündel in die Datenbank einfügt, um Schreibaufwand zu minimieren.

Das Einpflegen der Daten verläuft bei InfluxDB leicht anders. Da InfluxDB, wie vom Entwickler empfohlen, unter Windows in einem Docker-Container läuft, verwenden wir zum Schreiben der Daten in den Container die zur Verfügung gestellte Http-API. Diese wird in der Python-Bibliothek im bereits erstellten API-Objekt zur Verfügung gestellt.

Es gibt nun mehrere Best-Practice Möglichkeiten mit Hilfe dieses Objekts die Datensätze zu übermitteln. Hierbei handelt es sich einmal um das InfluxDB Line Protocol, ein String, in welchem die Werte durch Leerzeichen separiert sind und Aufzählungen durch Kommata gekennzeichnet werden, eine Batch Sequenz, bei der mehrere der vorher genannten Strings als Array übermittelt werden, also eine Art Transaktion, und das Datapoint-Objekt, welches in dieser Seminararbeit verwendet wurde. Hier wird ein Datenpunkt-Objekt erstellt. Diesem Datenpunkt können alle nötigen Attribute über Setter-Funktionen übergeben werden. Diese können auch über den Punkt-Operator

aneinandergehängt werden. Der erstellte Datenpunkt erhält im Konstruktor den Namen der Messung und über einen Setter den Typ des Messwertes und den Wert des Messwertes.

Der jetzt vollständige Datenpunkt kann dann über das API-Objekt mit Hilfe der Write-Operation in unseren Bucket geschrieben werden. Ein abschließendes Commit, wie bei der MySQL, ist nicht nötig.

```
while True:
    cpuusage = psutil.cpu_percent(interval=0.1)

    p = influxdb_client.Point("cpu").field("cpu_usage", float(cpuusage))

    val = (str(1), str(cpuusage), str(1), str(datetime.datetime.now()))
    write_api.write(bucket, org, p)
    mycursor.execute(sql, val)
    mydb.commit()
```

Abbildung 3: Generierung der Messwerte mit psutil und Schreiben der Werte in jeweilige Datenbank

5 Queries

Um mit in Datenbanken eingefügten Daten arbeiten zu können, verwendet man häufig Queries. Queries sind in einer datenbanksystemspezifischen Abfragesprache geschriebene Abfragen, zur Suche von Information. Da das Ergebnis einer solchen Abfrage eine Teilmenge des Gesamtdatenbestandes ist, kann man Queries auch als Filter ansehen.

MySQL verwendet als Abfragesprache SQL („Structured Query Lanuage“). SQL ist eine weit verbreitete Data-Query-Language (DQL).

Im Gegensatz zu MySQL verwendet InfluxDB 2.0 eine andere DQL. InfluxDB bietet auch Unterstützung für eine SQL-Like Abfragesprache, InfluxQL. Diese kann über einen Endpunkt in der API auch noch in der Version 2.0 verwendet werden. In dieser Seminararbeit wurden Abfragen in MySQL und auch InfluxDB allerdings über die UI der jeweiligen Clients gemacht. Die Web-UI der InfluxDB ermöglicht nur Abragen in der von InfluxData eigens enwickelten Abfragesprache Flux.

5.1 Flux

Flux ist eine von InfluxData eigens entwickelte Sprache zum Abfragen von Daten. Sie ist optimiert für die Arbeit mit Zeitreihendaten. Um einen leichten Einstieg in die Programmierung mit Flux zu ermöglichen, ist die Sprachen stark an JavaScript angelehnt, eine der beliebtesten Skriptsprachen der Letzen Jahre. [14, 15, 16]

Wichtiges Grundkonzept der Sprache ist beispielsweise der zentrale Operator, der sogenannte „Pipe-Forward-Operator“: „|>“

Dieser Operator ermöglicht es in Flux verschiedene Operation aneinander zu ketten und ersetzt damit einen Punkt-Operator. Dies entzerrt den Code und macht ihn lesbarer. Nach jeder Operation können die Ergebnistabellen durch den Operator in eine weiter Operation geleitet werden, um weiter manipuliert oder anderweitig verarbeitet zu werden.

Außerdem arbeitet Flux in seinen Ergebnissen mit Tabellen. Die in einem Bucket abgelegten Daten werden also nach einer Abfrage in Tabellen strukturiert und zurückgegeben.

Mit Flux lässt sich auch grundlegende Arithmetik ausführen. Flux bietet den vollen Umfang einer SQL-Like DQL und ist dabei dafür optimiert, auf hochfrequenten Zeitreihendaten zu arbeiten.[3]

5.2 Beständige Abfragen

Continuous Queries („Beständige Abfragen“) sind praktische Hilfsmittel zur langlaufenden Überwachung von Prozessen und werden daher hier im speziellen hervorgehoben. Sie ermöglichen es in regelmäßigen Abständen automatisiert vorher definierte Abfragen zu tätigen.

Unter InfluxDB wird dies in der neusten Version über Tasks geregelt. Ein Task ist hier ein Objekt, welches nach einen definierten Zeitplan Abfragen ausführt.

Dazu erhält ein Task zunächst einige definierende Informationen, die „Task Options“. Hier werden also grundlegende Dinge wie Name und Zeitplan definiert. Ein Zeitplan legt fest, wann und wie häufig ein Task läuft. Dazu muss ein Zeitintervall angegeben werden, entweder in Sekunden/Minuten/Stunden/… oder als Cron-Daemon. Es kann ein Offset angegeben werden, der die Ausführung des Tasks um das angegebene Intervall verzögert, aber das ursprüngliche Abfrageintervall beibehält, um beispielsweise verspätete Werte mit zu berücksichtigen. Es kann auch noch angegeben werden, wie viele Tasks gleichzeitig laufen können und nach wie vielen gescheiterten Versuchen ein Task als fehlgeschlagen angesehen wird.

Der zweite wichtige Teil eines Tasks ist die Abfrage selbst, die der Task ausführen soll. Hier kann man unter InfluxDB eine beliebige Query, zum Beispiel geschrieben in Flux, verwenden. Nachdem ein Task definiert und gespeichert ist, kann er auch beliebig ein- und ausgeschaltet werden. [6]

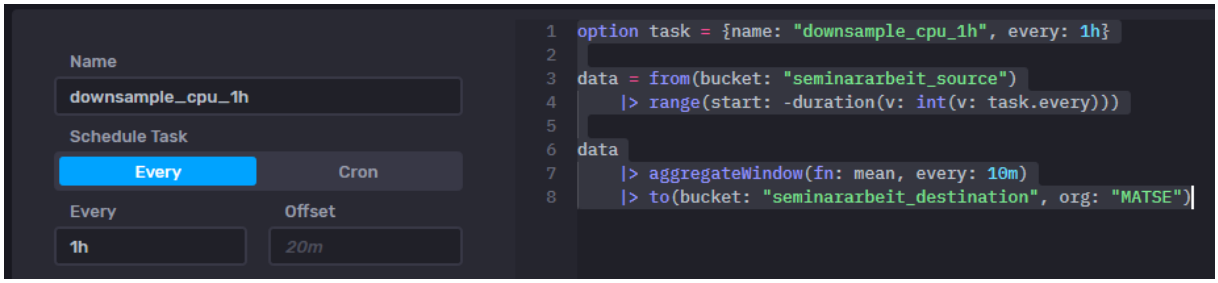

Abbildung 4: Task im Webinterface von InfluxDB, geschrieben in Flux

Ein ähnliches Konzept lässt sich auch in MySQL umsetzen. Hier heißt das genutzt Objekt „Event“ und wird zentral über den Event-Scheduler gesteuert.

```
CREATE EVENT testtaskt
ON SCHEDULE EVERY 1 HOUR
DO
    SELECT * FROM seminararbeit.data
```

Abbildung 5: Beispielevent in MySQL, welches jede Stunde für einen unbegrenzten Zeitraum ausgeführt wird

Ein Event in MySQL ist ähnlich strukturiert wie ein Task in InfluxDB. Es besitzt auch einen eindeutigen Namen, einen Zeitplan der Ausführung und eine Abfrage, welche zu gegebenem Zeitpunkt ausgeführt wird.

Events in MySQL können über Angabe eines Zeitplans gesteuert werden. Man kann ein Wiederholungsintervall angeben, sowie einen Start- und Endzeitpunkt. Lässt man diese Angabe weg und gibt stattdessen nur einen Zeitstempel an, so wird das Event nur einmal zum angegebenen Zeitpunkt ausgeführt.

Nach Ausführung eines Events oder nach Ablauf des Ausführungszeitraums wird dieses standartmäßig gelöscht. Das kann verhindert werden, wenn man angibt, dass es bei erfolgreicher Ausführung beibehalten werden soll („ON COMPLETION RESERVE“).

Wenn der Zeitplan und das Ausführungsverhalten definiert sind, folgt der Abfrageteil des Events. Hier kann mit normalen SQL-Statements gearbeitet werden. [17,18]

6 Speicherung

Es handelt sich bei InfluxDB und bei MySQL um grundlegend verschiedene Datenbanksysteme. Dies spiegelt sich auch in der Speicherung der Daten wider. In diesem Kapitel werden Unterschiede in der Indizierung von Datensätzen, Speicherverhalten und Möglichkeiten der Datenreduzierung herausgestellt.

MySQL, als relationales Datenbanksystem, speichert die Daten in Tabellen, welche durch Fremdschlüsseln relational voneinander abhängig sind. Gespeicherte Daten können hier durch Indizes in den Tabellen identifiziert werden. In diesen Tabellen sollten Indizes eindeutig sein, damit kein Datenverlust durch Dopplung eintritt.

Für die physische Abspeicherung der Daten auf dem Speichermedium bietet MySQL die Möglichkeit, aus verschiedenen Storage Engines auszuwählen.

Eine Storage Engine ist die Komponente, welche die physische Abspeicherung und das Lesen der Daten vom Speichermedium regelt. Hier gibt es für Anwendungsfälle optimierte Engines. In dieser Seminararbeit wurde InnoDB verwendet, seit MySQL 5.5 die Standard Storage Engine.

InfluxDB ist eine No-SQL Timeseries-Datenbank und speichert die Daten anders als beispielsweise MySQL. Unter InfluxDB werden Datenpunkte Labels, zusätzliche Tags und den Zeitstempel des Messwertes identifiziert. Sollte hier eine Dopplung vorliegen könnten beispielsweise Tags hinzugefügt werden oder der Zeitstempel um eine Nanosekunde verschoben werden.

InfluxDB nutzt ihre eigene Storage Engine, welche ein WAL („Write Ahead Log“), den Arbeitsspeicher, einen Time-Structured Merge Tree und einen Time Series Index verwendet. Hervorzuheben ist hier der Time Series Index. Er ermöglicht es sicherzustellen, dass auch mit über die Zeit anwachsenden Schlüsselsets gestellte Anfragen immer noch schnell bearbeitet werden können. [5, 13, 23]

Der genutzte Speicherplatz der MySQL-Datenbank lässt sich durch eine einfache Query abfragen. Nach ca. 24 stündiger Laufzeit des Skripts beträgt dieser ungefähr 41,1 MB. Eine ähnliche simple Möglichkeit den Speicherverbrauch des verwendeten InfluxDB-Buckets zu erhalten, ist nicht möglich. InfluxData stellt weder über ihre http-API noch über ihr Command Line Interface eine Möglichkeit zur Verfügung, dies abzufragen. Eine möglicher Lösungsansatz wäre die der Zugriff auf die Dateistruktur innerhalb des Docker Containers, um darüber die Größe der Datenbank zu ermitteln. Auch das würde aber im kurzen Beobachtungszeitraum der Seminararbeit schwierig sein, da die Daten in einem Bucket zunächst im Arbeitsspeicher gespeichert sind und damit ihre Größe nicht einfach auszulesen ist.

6.1 Aufbewahrungsrichtlinien

Speicher und Speicherzugriff kann in Datenbanksystemen bereits teilweise über die verwendete Storage Engine geregelt werden. Dennoch ist eine Speicherreduzierung bei langlaufenden Untersuchungen durchaus sinnvoll. Hier bieten sich Aufbewahrungsrichtlinien, sogenannte „Retention Policies“ an.

Eine Retention Policy gibt an, wie lange Daten vorzuhalten sind, bevor sie gelöscht werden können. Das klingt zunächst radikal, gerade im Bereich Datenbanken. Retention Policies können aber ein mächtiges Werkzeug sein, wenn es darum geht, Speicherplatzknappheit entgegen zu wirken.

In InfluxDB sind Retention Policies bereits integriert. Zusammen mit einer Datenbank bilden sie einen Bucket. Ein Bucket verfügt also immer über mindestens eine Retention Policy. Beim Einfügen von Daten in einen Bucket muss jetzt eine dem Bucket zugehörige Retention Policy angegeben werden. Wird dies nicht getan, wird der Wert automatisch in die Standard Retention Policy des Buckets eingefügt. Diese besagt, soweit nicht anders definiert, dass kein Wert je gelöscht wird. [4]

Eine solche Mechanik ist in der MySQL nicht nativ implementiert, kann aber über Umwege nachgebaut werden. Hierzu würden sich Events eignen, die in der Arbeit bereits erläutert wurden:

Nehmen wir an, wir haben eine nicht näher spezifizierte Tabelle, in welcher jeder Eintrag einen Zeitstempel besitzt. Die Daten sollen in dieser Tabelle nur ein Stunde lang gespeichert werden.

Mit diesen Vorgaben können wir also nun ein Event definieren, welches jede Stunde ausgeführt wird. Nun gibt es mehrere Möglichkeiten zu Löschende Werte zu identifizieren: Es könnten beispielsweise zunächst in einem geschachtelten SQL-DELETE-Statement alle Werte mit einem Zeitstempel kleiner dem aktuellen Zeitstempel gefiltert und dann gelöscht werden. So haben wir also mit der Hilfe der Events in MySQL eine Art Retention Policy erstellt, welche Daten aus unserer Tabelle nach einer Stunde löscht.

Daten nach einer vordefinierten Zeit zu löschen führt natürlich zur Speicherreduzierung, allerdings ist es wahrscheinlich in den meisten Situation und Andwendungsszenarien nicht sinnvoll Daten nur für einen kurzen Zeitraum zu speichern. Will man dennoch Speicherplatz reduzieren, gibt es noch eine weitere einfache Möglichkeit.

6.2 Downsampling

Das unterliegende Konzept von Downsampling ist es die Anzahl von Stützstellen, beispielsweise in Zeitreihen, zu reduzieren. Damit ist Downsampling klar mit Datenverlust verbunden! Dennoch ist es bei einer Langzeituntersuchung anstrebenswert.

Downsampling kann in InfluxDB und MySQL aus einem Zusammenspiel von zwei bereits genannten Komponenten erreicht werden: Continuous Queries und Retention Policies.

Eine Lösung, auf die im Folgenden eingegangen wird, verwendet zwei Tabellen: Eine Ausgangstabelle, in welche die Daten direkt von der Quelle eingefügt werden und eine Zieltabelle, in welche die Downsampling-Daten geschrieben werden. Zumindest die Ausgangstabelle besitzt eine Retention Policy. Sollen auch die Downsampling-Daten nach einer bestimmten Zeit gelöscht werden, bekommt sie auch eine Retention Policy, welche aber länger als die der Ausgangstabelle sein sollte.

Auf der Ausgangstabelle läuft zudem eine beständige Query, oder im Fall der InfluxDB ein Task beziehungsweise bei der MySQL ein Event. In dieser beständigen Query werden die Werte gefiltert, die zu Downsamplen sind. Die Werte werden anhand des Zeitstempel in Intervalle eingeteilt, über welche dann beispielsweise der Durchschnitt berechnet werden kann. Diese Werte sind also nun zusammengefasst und können in die Zieltabelle eingefügt werden. Die hochaufgelösten Daten in der Ausgangstabelle werden nun von der Retention Policy gelöscht. [7]

Es kann sinnvoll sein, die hochauflösenden Daten noch etwas gespeichert zu lassen, um diese auch untersuchen zu können.

Für die Umsetzung des Downsampling in MySQL und InfluxDB wurde bereits beschrieben, wie eine Retention Policy und ein Task oder Event erstellt werden kann.

Nach dem Downsampling der gemessenen Daten wurde der Speicherverbrauch der jeweiligen Datenbanken verglichen:

Für die InfluxDB ist genau wie die vorherige Speicherplatzauswertung dies leider nicht möglich. Datenreduktion ist allerdings ein immersiver Teil von InfluxDB und somit kann davon ausgegangen werden, dass Dies stattgefunden hat, wenn auch nicht messbar. In der MySQL lässt sich allerdings eine deutliche Speicherplatzeinsparung feststellen. Messwerte wurden in der Ausgangstabelle nur 12 Stunden vorgehalten. Jede Stunde wurden die Daten der letzten Stunde in 10-Minuten-Intervalle unterteilt und über jedes Intervall jeweils der Durchschnitt berechnet. Diese Werte wurden dann in die Zieltabelle geschrieben. Die Gesamtgröße der Datenbank beträgt nun anstelle der Ursprünglichen 41,1 GB nur noch ca. 22,558 GB. Diese Zahl setzt sich aus der Größe der Ausgangsdatenbank mit ca. 22,55 GB und der Größe der Zieldatenbank mit ungefähr 8 KB zusammen. Eine deutliche Reduktion in Speicherplatz.

7 Vergleich

Im Laufe der Seminararbeit wurden bereits Unterschiede im Hinblick auf Arbeitsfluss, Dateneinpflegung, Datenabfrage und Speicherkonzepten angedeutet, was hier ausführlicher betrachtet werden soll.

Das Timeseries-Datenbanksystem InfluxDB und das relationale Datenbanksystem MySQL unterscheiden sich bereits in ihrem Anwendungsbereich. InfluxDB ist, als für Zeitreihendaten optimiertes Datenbanksystem, auf den in unserem Ausgangszenario beschriebenen Anwendungsfall ausgelegt. Hochfrequent anfallende Daten unterschiedlichen Aufbaus können einfach und kostengünstig an einer Stelle gesammelt, gefiltert, und analysiert werden. Das Schema der Datenbank ist am Anfang egal, lediglich über eine Kennzeichnung mittels Flags und Labels muss sich Gedanken gemacht werden. Dadurch ist InfluxDB sehr flexibel. Messwerte können also mit einer niedrigen Einstiegsbarriere in die Datenbank geschrieben werden.

MySQL hat als relationales Datenbanksystem einen weit größeren Anwendungsbereich als InfluxDB, umfasst deren Anwendungsbereich sogar in ihrer eigenen Funktionalität. Durch die feste Tabellenstruktur muss sich über alle benötigten Attribute bereits im Vorhinein Gedanken gemacht worden sein. Eine nachträgliche Erweiterung einer Relation ist nur schwer möglich. Das schränkt die Flexibilität bei beispielsweise neu dazugewonnen Messungen, Messwerten, oder messbaren Attributen deutlich ein. Man könnte dies durch weitere Relation umgehen, was allerdings wiederum die anzustrebende Normalisierung der Datenbank umwirft.

Auch das eigentliche Einpflegen der Messwerte kann noch einmal genauer betrachtet werden. Es wurde gezeigt, wie die Messwerte aus den Python-Skripten in die jeweiligen Datenbanken eingepflegt werden. Der InfluxDB-Client stellt die Http-API zur Verfügung, welche die Daten der InfluxDB-Instanz übergibt. Jede Funktionalität, welche über die API genutzt werden kann, bekommt eine eigene Funktion. Durch die Http-API ist InfluxDB an vielen Stellen leicht anzuwenden und es lassen sich selbst von schwachen Mikrokontrollern und Sensoren direkt Daten einfügen. Die Vielzahl an Endpunkten, die zum Einfügen von Daten genutzt werden können, bieten Einsteigern die Möglichkeit über einen präferierten Endpunkt einfach und schnell Daten einzufügen.

Dem gegenüber steht der MySQL-Client, welcher eine zentrale Möglichkeit bietet, Befehle auf der Datenbank über den Datenbankcursor auszuführen. Hier können alle möglichen SQL-Befehle ausgeführt werden. Das biete Entwicklern und Anwendern, welche bereits Erfahrung mit SQL haben, eine einfache Möglichkaut auf der Datenbank zu arbeiten.

Die Unterstützung von SQL ist ein weiterer wichtiger Punkt im Vergleich der beiden Datenbanksysteme. SQL ist die am weitest verbreitete Query Language und die Grundlage

relationaler Datenbanksysteme. Durch diesen Status gibt es bereits einen großen Anwender- und Entwicklerkreis, welcher mit SQL arbeitet. Sie wird zu dem auch von den meisten Datenbanksystemen unterstütz. Das ist ein wichtiger Punkt im Hinblick auf Kompatibilität, wenn zum Beispiel zu einem anderen Datenbanksystem gewechselt wird, sollten sich beispielsweise Lizenz Bedingungen ändern.

Auch InfluxDB unterstützt eine SQL-Like Anfragensprache. Version 2.0 ist allerdings auf die Verwendung der hauseigenen Sprache Flux ausgelegt. Was Flux ist und wie sie grob aufgebaut ist, wurde bereits in Kapitel 5 gezeigt. Die Integration der eigenen Anfragensprache ermöglicht InfluxData Features der InfluxDB enger mit der Sprache zu Verbinden und zusätzliche Features, welche vom SQL-Standard nicht unterstützt werden, hinzuzufügen.

Eine Problemlösung geschrieben in Flux und unter Verwendung von InfluxDB kann optimal für einen gegeben Anwendungsfall sein, aber auch hier stellt sich die gleiche Frage wie zuvor mit MySQL. Sollten sich beispielsweise die Lizenzbedingungen der InfluxDB für Unternehmen ändern oder es muss aus einem anderen Grund auf ein anderes Datenbanksystem umgestiegen werden, so könne sich Probleme ergeben. Das neue Datenbanksystem verwendet wahrscheinlich nicht Flux sondern SQL. Eine wie oben beschriebene Problemlösung kann unter Umständen nicht ohne hohen Arbeits- und Resourcenaufwand umgesetzt werden. Dieser Punkt ist definitiv nicht zu vernachlässigen.

Auch in den Möglichkeiten der Speicherung der Messdaten unterscheiden sich die beiden Datenbanksysteme. Da InfluxDB auf hochfrequente Zeitreihendaten und Monitoring ausgelegt ist, finden sich hier bereits integrierte Features wie beispielsweise Retention Policies wieder. Diese sind fest in den Arbeitsfluss von InfluxDB integriert. In der Seminararbeit wurde gezeigt, dass diese Features zwar nicht direkt in der MySQL integriert sind, allerdings über Umwege nachimplementiert werden könnten.

Es wurde gezeigt, dass sich durch Downsampling eine deutliche, zumindest in der MySQL messbare, Datenreduzierung erzielen lässt. Natürlich ist dies auch mit Datenverlust verbunden, der ist hier aber im Sinne der Speicherreduzierung gewollt ist.

Wie also eignen sich die beiden Datenbanksysteme MySQL und InfluxDB für das am Anfang beschriebene Ausgangsszenario: Batterieprüfstände; und sind die Lösungen skalierbar?

Beide Datenbankmanagementsysteme weisen Vor- und Nachteile auf, welche in diesem Kapitel bereits näher beschrieben wurden. Das am Anfang beschriebene Ausgangsszenario geht aber nun nicht von einer einzigen Datenquelle aus, sondern beschreibt eine Vielzahl von Prüfständen. Wie gut lassen sich also beide Lösungsansätze skalieren.

InfluxDB ermöglicht eine einfache Erweiterbarkeit durch das Erstellen eines neuen Buckets. In diese Bucket können nun Daten eines weiteren Prüfstandes eingefügt werden. Je nach Anzahl der Prüfstände stößt auch diese Lösung schnell an seine Grenzen. Eine andere Möglichkeit wäre es weiter nur einen Bucket zu nutzen und neu Daten mit weiteren Labels und Tags zu versehen, um Zugehörigkeit zu verschiedenen Prüfständen zu zeigen. Die erhöhte Datenlast beim Schreiben der Daten in eine Bucket stellt für InfluxDB keine großen Probleme da, da diese auf eine große Zahl an Datenströmen ausgelegt ist.

Auch MySQL würde eine Erweiterung mit Hilfe von weiteren Relationen ermöglichen, was aber umständlicher als das Erweitern um einen InfluxDB-Bucket ist. Auch das Schreiben aus mehreren Datenquellen auf eine oder wenige weitere Relationen ist umsetzbar, aber nicht über lange Zeit vertretbar, da MySQL auf eine solche Datenlast nicht ausgelegt ist. Das würde sich umgehen lassen, beispielsweise indem entweder mit Transaktionen gearbeitet wird oder mit einer Art Puffer, beispielsweise einer zweiten Datenbank, welche Daten im Arbeitsspeicher hinterlegt und dann in Paketen weiterleitet.

Die Aspekte, an welchen die beiden Datenbanksysteme oben verglichen wurden, lassen sich auch als Bewertungskriterien definieren: Arbeit mit hochfrequenten Messdaten, Handhabung, Speicherverbrauch, Speicherreduzierung, Kompatibilität und Skalierbarkeit.

8 Fazit

Die beiden Datenbanksysteme, welche im Rahmen dieser Seminararbeit im Hinblick auf Ablage von Messdaten hin näher untersucht wurden, weisen beide Vor- und Nachteile auf, welche im vorherigen Kapitel beschrieben wurden.

Mit beiden Datenbanksystemen lässt sich ein Arbeitsfluss erstellen, welcher das Anfangsszenario umsetzten kann. Anhand der oben definierten Bewertungskriterien überzeugt InfluxDB beim Arbeiten mit hochfrequenten Daten und der einfachen Handhabung im Bereich der Datenakquirierung und Verwaltung. Ein Vergleich des Speicherverbrauchs war es leider nicht möglich, InfluxDB zu bewerten, aber eine integrierte Möglichkeit der Speicherreduzierung kann trotzdem mit in die Bewertung einfließen. Durch die direkte Implementierung von Tasks und Retention Policies ist dies einfach möglich. Eine Lösung mit InfluxDB lässt sich auch einfach skalieren, wie oben beschreiben wurde. Abstriche müssen bei InfluxDB im Bereich der Kompatibilität gemacht werden, da die neuste Version der InfluxDB vermehrt auf die Nutzung der eigenen Data Query Language, Flux, setzt.

MySQL bietet auch eine einfache Handhabung, auch wenn mehr Vorwissen im Bezug auf Datenbankarchitektur vorausgesetzt wird. Auch kommt MySQL gut mit den hochfrequenten Messdaten zurecht, wobei hier Bedenken bei einer möglichen Skalierung aufkommen, da MySQL nicht für solche Massen an Daten optimiert ist. Speicherreduzierung funktioniert durch ein nachgebautes Downsamplingverhalten auch gut. MySQL und dafür geschriebene Anfragen sind durch die Verwendung von SQL als Abfragesprache kompatibel mit einer weit reichenden Auswahl anderer Datenbanksysteme.

Betrachtet man diese Bewertung lässt sich folgendes Fazit im Bezug auf den Vergleich eines relationalen Datenbanksystems mit einem Time-Series-Datenbanksystems zur Ablage von Messdaten ziehen:

Die Seminararbeit hat herausgestellt, dass eine Lösung mit InfluxDB als Datenbankmanagementsystem als für den Anwendungsfall optimiertes Datenbanksystem gut geeignet für den Umgang mit Messdaten ist. Auch MySQL bietet weitreichende Möglichkeiten in diesem Bereich, reicht aber nicht an das Feature Set von InfluxDB in diesem Bereich heran.

9 Ausblick

In dieser Seminararbeit wurden zwei Datenbanksysteme im Hinblick auf Ablage von Messwerten verglichen. Es wurden auch Untersuchungen zum Speicherplatzverbrauch der Datenbanksysteme unternommen. Weitere Untersuchungen in der Zukunft könnten sich mit der Performance eines oder beider Systeme in Bezug auf Datendurchsatz und Schreibgeschwindigkeit beschäftigen. Auch eine tiefere Analyse der Speichernutzung der Datenbanksysteme mit größeren Datenmengen ist sinnvoll. Beide Systeme würden sich eignen, um Untersuchungen im Bereich Monitoring durchzuführen.

Mit den Ergebnissen dieser Arbeit und weiteren, zukünftigen, Untersuchungen wäre die Erarbeitung und Implementation einer Gesamtlösung des Ausgangsszenarios denkbar. Die Ergebnisse bieten auch Grundlage für weitere Problemlösungen mit ähnlichen Anforderungen, welche in zukünftigen Arbeiten behandelt werden könnten.

10 Quellenverzeichnis

[1] https://www.influxdata.com/products/influxdb/

[2] https://docs.influxdata.com/influxdb/v2.0/

[3] https://docs.influxdata.com/influxdb/v2.0/query-data/get-started/query-influxdb/

[4] https://docs.influxdata.com/influxdb/v2.0/reference/api/influxdb-1x/dbrp/

[5] https://docs.influxdata.com/influxdb/v2.0/reference/internals/storage-engine/

[6] https://docs.influxdata.com/influxdb/v2.0/process-data/get-started/#define-task-options

[7] https://www.influxdata.com/blog/downsampling-influxdb-v2-0/

[8] https://docs.influxdata.com/influxdb/v2.0/reference/internals/storage-engine/

[9] https://docs.influxdata.com/influxdb/v2.0/write-data/best-practices/duplicate-points/#increment-the-timestamp

[10] https://db-engines.com/en/system/InfluxDB#a32

[11] https://db-engines.com/en/article/Time+Series+DBMS

[12] https://db-engines.com/de/system/MySQL

[13] https://db-engines.com/de/article/Storage+Engine

[14] https://insights.stackoverflow.com/survey/2020#most-popular-technologies

[15] https://insights.stackoverflow.com/survey/2019#most-popular-technologies

[16] https://insights.stackoverflow.com/survey/2018#most-popular-technologies

[17] https://dev.mysql.com/doc/refman/8.0/en/events-overview.html

[18] https://www.mysqltutorial.org/mysql-triggers/working-mysql-scheduled-event/

[19] https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursor.html

[20] https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysql-connector.html

[21] https://de.wikipedia.org/wiki/Abfragesprache

[22] https://de.wikipedia.org/wiki/Messwert

[23] https://www.w3resource.com/mysql/mysql-storage-engines.php

[24] https://pypi.org/project/psutil/

[25] https://db-engines.com/de/ranking

[26] https://db-engines.com/en/ranking/time+series+dbms

[27] https://www.mysql.com/de/

[28] https://www.influxdata.com/

[29] https://www.influxdata.com/blog/getting-started-with-python-and-influxdb-v2-0/

Alle Quellen sind Stand: 13.12.2020