Oliver A. Tazl, BSc

SQL vs NoSQL

Analyse und Vergleich der Performance von relationalen Datenbanksystemen und strukturierten Datenspeichern in verteilten Umgebungen in Hinblick auf Big Data

Technische Universität Graz

Institut für Softwaretechnologie

Head: Univ.-Prof. Dipl-Ing. Dr. techn. Wolfgang Slany

Supervisor: Univ.-Prof. Dipl.-Ing. Dr. techn. Franz Wotawa

Graz, Februar 2017

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Date Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Datum Unterschrift

Danksagungen

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Masterarbeit sowie meines Studiums unterstützt und motiviert haben.

Zuallererst möchte ich Herrn *Univ.-Prof. Dr. Franz Wotawa* für die fachlichen und persönlichen Support danken. *Uptime GmbH* als Forschungspartner und Förderer dieser Masterarbeit sei ebenfalls dankend erwähnt.

Darüber hinaus möchte ich mich bei meinen Eltern, *Heimo* und *Barbara Tazl*, und Großmüttern, *Erna Rabinig* und *Katharina Koller*, bedanken, die mir durch ihre Unterstützung mein Studium ermöglicht haben. Ich danke euch von ganzen Herzen!

Auch Freunde und Wegbegleiter seien an der Stelle erwähnt. Besonders hervorheben möchte ich aus all diesen hier Geschäftsführer *Stefan Hausberger*, *Mag. Matthias Kaltenegger*, *Mag. Peter Pickl*, *Christopher Tafeit, BSc* und *Marco Tatschl, BA (econ.)*.

Abschließend möchte ich mich bei *Mag. phil. Martin Berger* für das Lektorat besonders bedanken.

Oliver Tazl,

Kurzfassung

In Zeiten wie diesen, in denen Datenmengen ständigem Wachstum unterliegen, ist es wichtig diese in mehr oder weniger strukturierter Form rasch zu speichern. Hierzu werden unterschiedlichste Systeme im Bereich der RDBMS und strukturierten Datenspeichern verwendet. Diese bieten unterschiedliche Eigenschaften, Vor- und Nachteile. Aspekte wie Geschwindigkeit und Möglichkeit der Datenspeicherung bzw. -auswertung stehen hier im Vordergrund.

Durch die Aufweichung verschiedener Grundprinzipien auf welche relationale Datenbanksysteme bauen, sind NoSQL-Systeme teilweise in der Lage große Vorteile im Bereich Geschwindigkeit zu bieten. Das zu optimierende Problem muss hierbei für jede Anwendung entsprechend neu bewertet und eine Abwägung der möglichen Optionen getroffen werden.

Diese Arbeit stellt mehrere Systeme unterschiedlicher Gruppen gegenüber um so Bild der Fähigkeiten und Möglichkeiten in diesen Bereichen zu analysieren und zu evaluieren.

Abstract

Amounts of data are rising continually, so it is more important than ever to store it fast in a more or less structured form. For this purpose there are many different kind of systems in the range of relational database management systems and structured data stores which could be used for that. These systems are offering a varity of features, characteristics, advantages and disadvantages. The main aspects are speed and possibilities of data storage, retrieval and evaluation.

By weakening the fundamental principles of RDBMS, NoSQL systems are able to get big improvments in speed. The optimal balance is a problem which has to be solved for every new application itself.

The main goals of this thesis is the comparison of multiple systems of different groups to get a evaluation of features and speed.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Code Sample-Verzeichnis

Abkürzungsverzeichnis

DBMS Datenbankmanagementsystem

RDBMS Relationales Datenbankmanagementsystem

SQL Structured Query Language

NoSQL Not only SQL

ACID Atomicity, Consistency, Isolation, Durability

BASE Basically Available, Soft state, Eventual Consistency

CouchDB Cluster of unreliable commodity hardware Data Base

JSON JavaScript Object Notation

HTTP Hypertext Transfer Protocol

API Application Programming Interface

JVM Java Virtual Machine

JDK Java Development Kit

RAM Random Access Memory

SSD Solid State Disk

N/A Not applicable - Nicht anwendbar

1. Einleitung

In Zeiten wie diesen, wo Datenmengen täglich steigen und es viele datenbasierte Anwendungen gibt, ist eine effiziente und rasche Speicherung ebenso wichtig wie der parallelen Zugriff von tausenden Nutzern.

Das Feld der Systeme zur Lösung dieser Anforderungen ist sehr breit. Relationale Datenbanksysteme auf der einen, strukturierte Datenspeicher auf der anderen Seite. Die Unterschiede und Gemeinsamkeiten, die Vor- und Nachteile im Bezug auf Performance der jeweiligen Systeme herauszuarbeiten ist Ziel dieser Arbeit. Es geht dabei sowohl um einen Performance- als auch um einen Funktionsvergleich der einzelnen Systeme. Der Geschwindigkeitsvorteil der NoSQL-Systeme wird meist durch eine schwache Struktur erkauft.

Auch im Bereich des Datenzugriffs bestehen wesentliche Unterschiede. Während die RDBMS über SQL als Abfrage- und Beschreibungssprache verfügen, wird bei strukturierten Datenspeichern meist ein anderer Weg beschritten. Von HTTP-APIs bis hin zu SQL-Subsets ist dort alles vertreten.

1.1. Was ist Big Data?

Big Data ist ein Begriff der allgegenwärtig ist. Aufgrund der Tatsache, dass dieser irgendwo zwischen Medien, Industrie und akademischer Community entstanden ist, fehlt leider eine eindeutige Definition. Am Ehesten definiert sich dieser Begriff über die Menge, die Komplexität und die Schnelllebigkeit der Daten. [24][8]

Daten werden in immer größerer Menge gesammelt und verarbeitet. Seien es Bewegungsprofile aus GPS-Daten, der Puls aus der Smartwatch oder Daten über Nahrungsaufnahme aus einer App: All diese Massendaten müssen meist schnell und skalierbar aufgezeichnet und verarbeitet werden könne. Maschinell erzeugte Daten, wie z.B. Logdateien und Verbindungsaufzeichnung beschleunigen die Steigerung der Datenmenge enorm. Auch individuelle Surfprofile der Nutzer fallen unter diese Gruppe. Sie finden vorallem Verwendung in der Online-Werbebranche, um Vorlieben der Nutzer auszuforschen und zielgerichteter Werbung platzieren zu können. Ein weiteres Anwendungsgebiet, welches mit Big Data abgedeckt werden kann, ist die zeitnahe Erkennung und unüblichen Verhalten. Dies kann einerseits in der Systemüberwachung eingesetzt werden, um Ausfälle vorherzusagen bevor sie wirklich passieren, andererseits kann es auch in der Betrugsbekämpfung eingesetzt werden um etwa Zahlungen mit gestohlenen Kreditkartendaten zu stoppen und damit Schaden von Unternehmen und Personen abzuwenden.[12]

Diese Art der Daten steht unter Kritik bei vielen Datenschützern, da mit ihnen ein Profil des Nutzers erstellt und somit seine Privatsphäre eingeschränkt wird bzw. werden kann. Daher ist es von großer Bedeutung mit solchen, meist personenbezogenen Daten mit besonderer Vorsicht umzugehen. [17]

Im Vergleich zu traditionellen Daten zeigt Big Data ein stark verändertes Verhalten. Nicht nur die Menge der Daten wächst enorm auch die Lebensdauer sinkt zum Teil. Während auf der einen Seite klassisch zentrale Datenbanken stehen, geht man bei Big Data hin zu stark verteilten Systemen. Dies is aufgrund der Tatsache notwendig geworden, da einzelne Systeme nicht mehr die Anforderungen an Speicher und Leistung gerecht werden können. Wie bereits vorher erwähnt steigen die generierten Datenraten von täglicher oder stündlicher Erfassung hin zu sekündlicher oder noch höher. Die Struktur der Daten nimmt ebenso weiter ab. Die Steigerungsraten der Datenmengen sind enorm und es wird immer schwieriger aus solchen Mengen an Informationen nutzbringende Inhalte zu extrahieren. [8]

1.1.1. Partitionierung und Sharding

Die Partitionierung, die Aufteilung von Daten auf verschiedene physische oder virtuelle Systeme eine für die meisten Datenspeicher sehr wichtige Technik um Redundanz, damit Ausfallssicherheit und Volumen zu bewältigen. Hierbei werden die Daten horizontal und vertikal auf verschiedene Systeme verteilt. Beim vertikalen Partitionieren der Daten werden eines z.B.

Kunden auf mehrere Systeme verteilt. Hierbei wird das Schema unterteilt und die einzelnen Teile auf mehrere physische Systeme verteilt. Es werden z.B. alle Adressen der Kunden auf einem eigenen Server gespeichert.[11]

Bei Sharding, auch horizontale Partitionierung, handelt es sich um eine Strategie der Datenaufteilung auf mehrere Datenbanksysteme mit gleichem Schema. Dabei werden die Daten aufgrund von zuvor gewählten Kriterien auf die Einzelsysteme verteilt. Dabei ist auf jedem Einzelsystem das gleiche Schema im Einsatz. Die Verteilung kann automatisch vom einzelnen System unterstützt werden, wie dies bei z.B. Redis der Fall ist, oder es muss manuell eingegriffen werden. Dieses Thema betrifft im allgemeinen sowohl relationale Datenbanken als auch NoSQL-Systeme.[11]

1.2. Ziele & Hintergründe

Das Ziel der Arbeit ist es, Datenbanksysteme für die Speicherung, Abfrage und Auswertung von Sensordaten zu evaluieren. Hierbei wird vorallem die Performance der einzelnen Systeme in den Vordergrund gestellt. Die Auswahl der Testkandidaten erfolgte durch eine Verbreitungs- und Potenzialanalyse. Es wurden sowohl kommerzielle als auch Open Source Software Projekte für die Tests herangezogen. Bei einigen Systemen wurde ebenso ein Vergleiche der kommerziellen und offenen Lösungen des selben Produktes durchgeführt.

Bei der Auswahl der zu evaluierenden Systeme kamen verschiedene Faktoren zum Einsatz. Hierbei wurde versucht etablierte Vertreter und aufstrebende Projekte zu kombinieren. Dabei war es vorallem wichtig Systeme zu wählen, deren Stabilität bereits einen gewissen Punkt überschritten haben, da Systeme, an deren Stabilität zu zweifeln ist eher nicht für den Langzeiteinsatz bzw. Produktiveinsatz verwendet werden können. Weiters war es bei der Auswahl ebenso bedeutsam verschiedene Vertreter der unterschiedlichen Gruppen vorallem im Bereich der NoSQL-Datenspeicher zu wählen.

Die Kandidaten aus der Gruppe der relationalen Datenbanken sind MySQL, PostgreSQL, Oracle DB und IBM DB2. Im Falle von MySQL wurden ebenfalls die Varianten von Percona, MariaDB und der Galera Cluster ausgewertet. Die Auswahl deckt alle großen Datenbankhersteller und -systeme in diesem Bereich ab.

Die Kandidaten der NoSQL-Systeme teilen sich in mehrere Untergruppen. Da diese Gruppe sehr viele verschiedene Vertreter aufweist musste hier stärker vorselektiert werden als bei den relationalen Datenbanksystemen. Es sind bei den System oft Zuordnungen zu mehreren Gruppen möglich.

Aus der Reihe der dokumentorientierten Datenbanken wurde MongoDB und CouchDB gewählt. Hierbei muss erwähnt werden, das MongoDB mit Abstand die meist verbreitetste dokumentorientierte Datenbank ist, CouchDB hat hier eher eine Außenseiterrolle (vgl. [4]).

Die Gruppe der spaltenorientierten Datenbanken wird von Apache Cassandra und von InfluxDB vertreten. Apache Cassandra wird in das Testfeld aufgrund der hohen Verbreitung aufgenommen, InfluxDB wegen der Verbindung der aus spaltenorientierter Datenbank und Zeitreihendatenbank.

Als Vertreter der Key-Value-Stores und -Caches wurden Redis und Hazelcast gewählt. Redis wurde aufgrund seiner hohen Verbreitung in das Testfeld aufgenommen, Hazelcast wegen der Fähigkeiten als nahtloser Caching-Layer.

In Gruppe der Zeitreihendatenbanken besteht aus InfluxDB, OpenTSDB und Riak TS, wobei InfluxDB bereits in der Gruppe der spaltenorientierten Datenbanken behandelt wird. OpenTSDB ergänzt das Testfeld aufgrund der zusätzlichen Eigenschaft als Hadoop/HBase-Aufsatz. RiakTS wird als Newcomer in das Feld aufgenommen.

Als konkretes Beispiel zum Vergleich dient die Speicherung von Messdaten aus einer unterschiedlich hohen Anzahl an Sensoren. Hierbei kann es sich um Telemetriedaten eines Servers, Messergebnisse aus einem Fahrzeug oder anderen messtechnisch relevanten Anwendungen handeln. Diese Form der Daten kommt in einer immer stärker vernetzten Infrastruktur, in der viele verschiedene Größen ständiger Überwachung bedürfen bzw. diese gewünscht ist, viel häufiger vor.

Die Datenstrukturen und -schemata wurden zur im Hinblick auf Vergleichbarkeit optimiert. Daher viel die Wahl auf möglichst einfache und gut

abbildbare Schemata. Natürlich kann es hier zu Einschränkungen aufgrund von Systemspezifika geben.

1.3. Testkonfigurationen

Die Experimente wurden mit einem HP ZBook 15 der 1. Generation mit Intel Core i7-4800MQ, 2.7 GHz, 32 GB RAM und 256 GB SSD durchgeführt. Als Betriebssystem diente Ubuntu 16.04 in der Desktop-Edition.

Bei der Erstellung der Datenstrukturen wurde darauf geachtet eine möglichst ähnliche und damit vergleichbare Konfiguration zu erreichen. Durch leichte Unterscheide der Datenbanken waren entsprechende Anpassungen der Client Software nötig. Auch bei der Serverkonfiguration wurde auf eine äquivalente Konfiguration der Systeme geachtet. Der Vergleich der Standardkonfigurationen erschien als eine faire Methode um keine Bevorzugung durchzuführen.

Als derzeit am weit verbreitetsten Sprachen der Welt wurde Java als Entwicklungssprache gewählt. Die Java Virtual Machine (JVM) wurde in der Version JDK 8u111 verwendet. Zur Erstellung und zur Ausführung des Testcodes wurde JetBrains IntelliJ eingesetzt. Als Build-Management-Tool wurde Apache Maven verwendet um das Handling der unterschiedlichen Libaries zu vereinfachen.

Neben der Testausführung wurde der Laptop zu keinen anderen Tätigkeiten herangezogen und unnötige Hintergrundprozesse gestoppt um eine möglichst geringe Beeinflussung durch das System zu erreichen.

Einschränkend ist hier festzuhalten, dass das Multithreading- bzw. Multiclient-Potenzial der einzelnen Systeme aufgrund der vorliegenden Testkonfiguration nicht ausgewertet wurde, da es hier zu zu großen Verfälschungen aufgrund der Mehrfachbelastung kommen konnte.

1.3.1. Docker

Als Container für die einzelnen Datenbanksysteme kam Docker zu Einsatz.

Es ermöglicht das Packaging von Softwaresystemen. Hierbei können alle für die Ausführung relevanten Teile, wie Softwaretools, Bibliotheken und andere Komponenten zusammen mit der jeweiligen Software in Container verpackt werden. Diese Container können danach auf jedem Docker-Host ausgeführt werden. Im Gegensatz zu virtuellen Maschinen kommt es bei der Docker Engine zu einem wesentlich geringeren Overhead. Es kommt nicht bei jedem Docker-Container ein eigenes Gastbetriebssystem zum Einsatz. Andere Vorteile wie die Isolation der einzelnen Containern wird allerdings gewährleistet, was einen Beitrag zur Sicherheit und zur einfacheren Entwicklung von Systemen leistet. Docker kann ebenso einen Beitrag zu Skalierbarkeit von Applikationen leisten, da mit wenig Aufwand weitere

Instanzen eines Containers gestartet und bei entsprechender Ausgestaltung des Systems Last verteilt werden kann.

Dadurch ermöglicht Docker, nach Erstkonfiguration, eine schnelle Herstellung der einzelnen Testkonfigurationen. Dazu wurden selbst entwickelte Docker-Images, sowie von Herstellern und Contributern verwendet. Die einzelnen Images wurden evaluiert um etwaige für die Tests relevante Konfigurationen zu bestimmen.

Jedes Virtualisierungssystem bringt *per se* einen gewissen Overhead mit sich, welcher die Leistungsmessung beeinflussen kann. Aufgrund dessen wurden einige Faktoren, welche leistungsmindernde Auswirkungen zeigen können umgangen. [6]

Die Abbildung 1.1 veranschaulicht den Aufbau der Konfiguration. Im oberen Teil der Grafik wird der Ablauf bei einer Konfiguration ohne Cluster gezeigt. Hierbei ist die direkte Verbindung zwischen Server und Client erkennbar. Dies kann durch die Dockerkonfiguration *–net=host* erreicht werden. Dadurch reduziert sich der Overhead in der Netzwerkkommunikation. Im unteren Bereich sieht man die Konfiguration eines Clusters. Dieser benötigt die Docker-Netzwerk-Bridge um eine ausreichende Trennung der einzelnen Nodes von einander zu erreichen um die Konfiguration problemlos ausführen zu können. Dabei entsteht ein Overhead, welcher sich allerdings in Grenzen hält. [13]

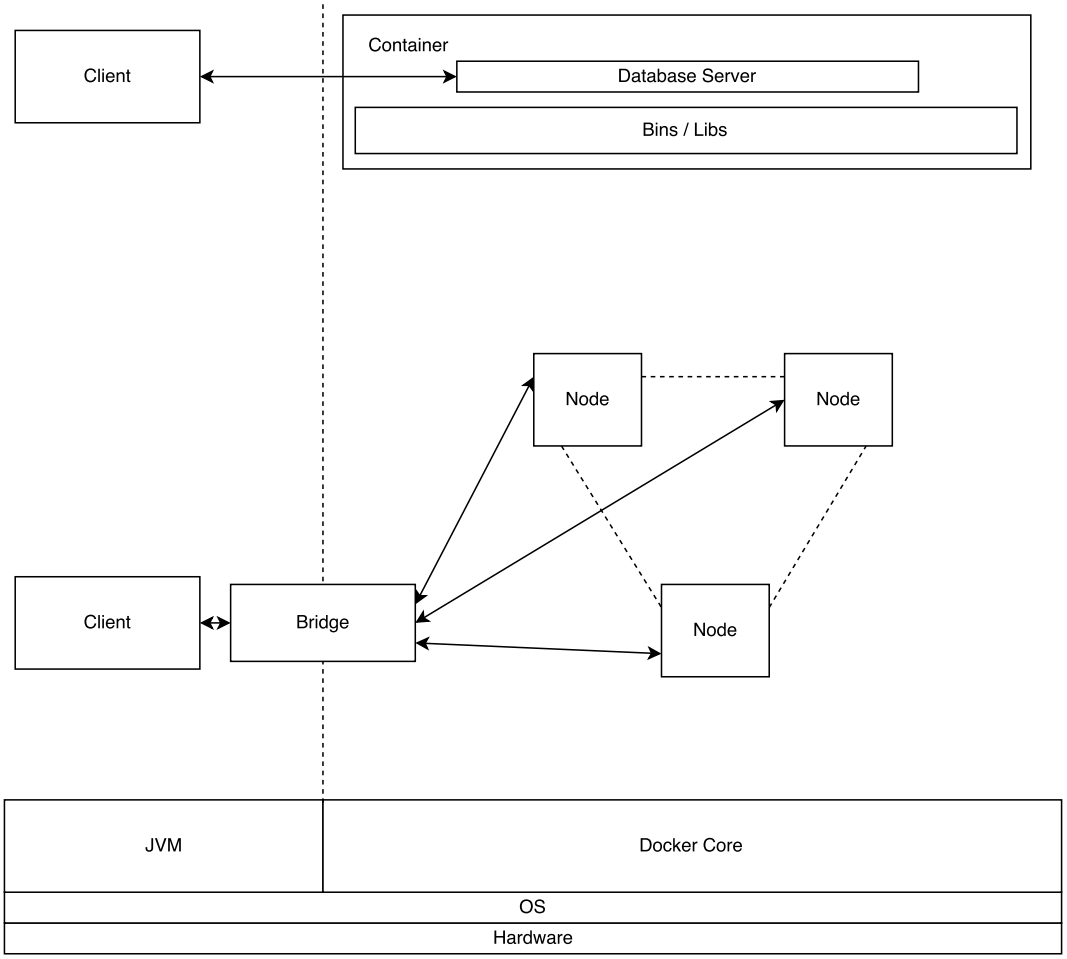

Abbildung 1.1.: Testkonfiguration mit Docker

2. Relationale Datenbanksysteme

Relationale Datenbanksysteme sind die Grundlage vieler Projekte und Systeme seit vielen Jahren und Jahrzehnten. In dieser Zeit standen sie immer neuen Anforderungen und Veränderungen gegenüber. Dieses Kapitel beschäftigt sich mit der Analyse der Performance der ausgewählten Testkandidaten.

2.1. Grundlagen

Relationales Datenbankmanagementsystem (RDBMS) beruhen auf einem tabellarischen relationalen Datenbankmodell, welches auf Edgar F. Codd [5] aus dem Jahre 1970 zurückzuführen ist. Als Datenbanksprache zur Abfrage und Veränderung dient dieser Art von System meist SQL. Die Daten werden in tabellarischer Form gespeichert und mit Schlüsseln in Beziehung gesetzt. Bei diesen Systemen kommt ebenfalls das ACID-Paradigma zum Einsatz.

2.1.1. ACID

ACID steht als Abkürzung für Atomarität, Konsistenzerhaltung (Consistency), Isolation und Dauerhaftigkeit und beschreibt ein grundlegendes Prinzip der relationalen Datenbanken. Relationale Datenbanksysteme versuchen sich stark an diesem Prinzip zu orientieren.

Atomarität

Eine atomare Operation charakterisiert die Tatsache, dass diese entweder vollständig oder nicht ausgeführt wird (All-or-Nothing). Transaktionen sind hier die Methode der Wahl. Die Operationen innerhalb einer solchen werden erst nach erfolgreichem Abschluss der letzten Operation in Kraft gesetzt. Kommt es während dieser Transaktion allerdings zu einem Fehler, wird diese komplett rückgängig gemacht. Es kommt zu einem Rollback.

Konsistenzerhaltung

Die Eigenschaft der Konsistenzerhaltung stellt sicher, dass eine Datenbank die vor einer Operation konsistent war, dies auch danach ist. Hierzu müssen natürlich Bedingungen für die Integrität der Daten definiert werden. Hierbei könnten z.B. Schlüssel- oder Fremdschlüsselbedingungen zum Einsatz kommen. Hiermit können selbstverständlich keine Fehler ausgeschlossen

werden, allerdings können diese nur innerhalb der definierten Parameter und Regeln passieren.

Isolation

Es kommt bei Datenbanken häufig dazu, dass Prozesse parallele Operationen innerhalb der Datenbank auslösen. Das Prinzip der Isolation soll verhindern bzw. weitgehend einschränken, dass diese sich gegenseitig beeinflussen. Je nach Grad der Isolation schlagen andere unvollständige Transaktionen auf das Ergebnis einer anderen Transaktion durch. Hierbei gibt es unterschiedliche Abstufungen.

Dauerhaftigkeit

Dauerhaftigkeit beinhaltet, dass nach Ende einer erfolgreichen Transaktion die Daten dauerhaft gespeichert sind. Auch ein Systemfehler oder -ausfall darf dies nicht beeinflussen. Ausfälle im Bereich des RAMs sind hier besonders hervorzuheben. Dies kann durch z.B. Transaktionslog garantiert werden. Es erlaubt dem DBMS die nach dem Ausfall angefallenen Operationen korrekt nachzuvollziehen.

2.1.2. Datenstruktur

Für den Geschwindigkeitsvergleich wurden zwei unterschiedliche Strukturen verwendet. Einerseits wurde eine der dritten Normalform entsprechende Tabelle zur Speicherung der Messdaten, andererseits eine mehrspaltige Variante, wobei jeder Sensor seine eigene Spalte erhält.

Variante 1

Die Variante 1 stellt sich in folgender Form dar: *SENSOR_VAL1(SID, TI, VALUE)*, wobei *SID* den Identifier für den Sensor, *TI* den entsprechenden Zeitstempel und *VALUE* den Sensorenwert enthält. Hierbei wird jeder Wert eines Sensors in eine Zeile der Tabelle gespeichert, was unterschiedliche Frequenzen bei der Datenanlieferung zulässt ohne die Datenmenge zu erhöhen.

Variante 2

Die Variante 2 gestaltet sich wie folgt: *SENSOR_VAL2(AID,TI, SENSOR1, SENSOR2,..., SENSOR p)*, wobei *AID* der Identifier für die Anlage und *TI* der entsprechende Zeitstempel ist. Die Werte jedes Sensors werden in einer eigens für diesen angelegten Spalte der Tabelle gespeichert. Diametral zu Variante 1 wird hier nicht auf Verfügbarkeit bzw. Granularität der Daten Rücksicht genommen. Dies führt zu einer erhöhten Datenmenge

bzw. zu Leereinträgen in der Tabelle. Auch fehlt bei dieser Variante jede Flexibilität in der Datenspeicherung. Jeder neu hinzukommende Sensor führt unweigerlich zu einer Änderung im Tabellendesign.

2.1.3. Transaktionssteuerung

Das Thema Transaktionssteuerung beeinflusst die Geschwindigkeit der Insertion-Queries in besonderem Maße. Aufgrund der Tatsache, dass nach jeder Transaktion ein dauerhafter Zustand durch das DBMS hergestellt werden muss, sind sehr häufig Festplatten bzw. SSD-Zugriffe nötig. Diese sind vergleichsweise sehr langsam und beeinflussen die Gesamtperformance stark.

Um die Unterschiede dieser Einzeltransaktionen im Vergleich zu Massentransaktionen herauszuarbeiten wurde bei der Implementierung eine Unterscheidung zwischen dem Verhalten „AutoCommit“ und „NoAutoCommit“ geschaffen. Hierbei wird bei dem jeweiligen JDBC-Treiber die entsprechende Einstellung angepasst. Dadurch ist ein Vergleich der automatischen und der manuellen Transaktionssteuerung möglich. Nachdem die AutoCommit-Option in der Praxis bei höheren Datenraten deaktiviert werden muss um einen entsprechenden Durchsatz zu erhalten, wurden die Tests mit aktiver Option auf zwei Testkandidaten eingeschränkt, MySQL und PostgreSQL.

2.1.4. Abfragen

Nach der Einfügungslaufzeit dient die Abfragelaufzeit als weiterer Indikator der Analyse. Bei der empirerischen Untersuchung werden hierbei die beiden Designvarianten einem Test unterzogen. Dazu wurden die 2 Tabellen *SENSOR_VAL1* und *SENSOR_VAL2*, wie oben bereits beschrieben in der jeweiligen Datenbank erzeugt und mit Werten befüllt. Es wurden für den Test die Daten von 100 Anlagen mit 100 Sensoren angenommen. Es wurden pro Sensor 20.160 Datensätze erzeugt, was insgesamt für *SENSOR_VAL1* 201.600.000 Tupel sowie für *SENSOR_VAL2* 2.016.000 Tupel ergibt. Unter der Prämisse, dass minütlich gemessen wird, sind das 2 Wochen an Daten.

Die erste Abfrage, *query1* bzw. *query1b*, bestimmt die Anzahl der in Tabelle gespeicherten Daten. Die beiden Varianten werden in Code Sample 2.1 dargestellt.

```
query1 = SELECT COUNT(*) FROM SENSOR\_VAL1;

query1b = SELECT COUNT(*) FROM SENSOR\_VAL2;
```

Code Sample 2.1: Abfrage der Anzahl der Tuple

In der zweite Abfrage, *query2* und *query2b* ersichtlich in Code Sample 2.2, werden alle Werte eines Sensors einer bestimmten Anlage ausgelesen.

```
query2 = SELECT * FROM SENSOR_VAL1 WHERE id=76 and aid = 42;
```

```
query2b = SELECT TI, VALUE76 FROM SENSOR_VAL2 WHERE aid = 42;
```

Code Sample 2.2: Abfrage der Daten eines Sensors

Die dritte Abfrage, *query3* und *query3b* ersichtlich in Code Sample 2.3, akquiriert die Daten einer Anlage innerhalb eines bestimmten Zeitfensters.

```
query3 = SELECT * FROM sensor_val1 WHERE TI > start AND TI< end
    AND aid = 23;

query3b = SELECT * FROM sensor_val2 WHERE TI > start AND TI < end
    AND aid = 23;
```

Code Sample 2.3: Abfrage der Daten einer spezifischen Zeitspanne

Die vierte Abfrage, *query4* bzw. *query4b* in Code Sample 2.4, bestimmt Minimum, Maximum und Durchschnitt eines Sensors einer definierten Anlage.

```
query4 = SELECT AVG(value), MIN(value), MAX(value) FROM
    sensor_val1 WHERE  id = 14 AND aid = 42;

query4b = SELECT AVG(value14), MIN(value14), MAX(value14) FROM
    sensor_val2 WHERE aid = 42;
```

Code Sample 2.4: Abfrage des Maximums des Minimums und des Durchschnitts eines Sensors

In der fünften Abfrage, *query5*, siehe Code Sample 2.5, werden Durchschnitt, Maximum und Minimum einer gesamten Anlage ermittelt. Diese Abfrage kann aufgrund der Struktur nicht auf SENSOR_VAL2 ausgeführt werden.

```
query5 = SELECT AVG(VALUE), MIN(VALUE), MAX(VALUE) FROM
    sensor_val1 WHERE AID='anlage45';
```

Code Sample 2.5: Query 5

Die sechste Abfrage, *query6*, siehe Code Sample 2.6, akquiriert die IDs der Sensoren, welche gleiche Werte auf eine und der selben Anlage liefern. Auch hier ist die Abfrage nur auf die Struktur von SENSOR_VAL1 anwendbar.

```
query6 = SELECT V1.id, V2.id FROM sensor_val1 V1, sensor_val1 V2
    WHERE V1.aid = 'anlage42' and V1.id<>V2.id AND V1.value=V2.
    value AND V1.ti=V2.ti;
```

Code Sample 2.6: Query 6

In der siebenten Abfrage, *query7* bzw. *query7b*, siehe Code Sample 2.7, werden die Zeitstempel ermittelt, welche den Beginn bzw. das Ende der verfügbaren Daten einer Anlage anzeigen.

```
query7 = SELECT MIN(ti), MAX(ti) FROM sensor_val1 WHERE aid = 65;

query7b = SELECT MIN(ti), MAX(ti) FROM sensor_val2 WHERE aid = 65;
```

Code Sample 2.7: Query 7

2.1.5. Implementierung

Bei der Implementierung wurde wie bereits erwähnt auf der JVM von Oracle durchgeführt. Als Sprache wurde Java 8 verwendet.

Grundsätzlich ist der Ablauf für einzelnen RDBMS sehr ähnlich aufgrund der Tatsache dass SQL von alle unterstützt und ein JDBC-Treiber vorliegt. Unterschiede tun sich nur in der Definition der einzelnen Datentype beim Erstellen der Tabellen und bei der Connection-URL der Datenbank auf. Hierzu wurden für die einzelnen Systeme Konfigurationen erstellt. Diese JSON-Dateien enthalten weiters Informationen zum Testablauf wie z.B. Tabellenname, maximale Anzahl der Sensoren, Intervall und Durchläufe pro Intervall (vgl. Code Sample 2.8).

```
{
        "connectionUrl": "jdbc:mysql://localhost/
           testdb?useSSL=false&serverTimezone=Europe/
           Vienna",
        "username": "user",
        "password": "******",
        "table": "testdb.sensor_val2",
        "dataTypeKey": "integer",
        "dataTypeTime": "bigint",
        "dataTypeData": "double",
        "autoCommit": true,
```

```
        "maxSensors": 200,
        "times": 10,
        "interval": 10,
        "insertionSamples": 1000,
        "cleanup": true,
        "create": true,
        "facility": 1
}
```

Code Sample 2.8: Konfigurationsfile MySQL Insert V2

Die einzelnen Clients erstellen einen csv-kompatiblen Output, welcher für die weiteren Auswertungen herangezogen wurde.

Für die Vorbereitung der Abfragetests wurde eigens eine Konfiguration, welche die entsprechenden Anforderungen laut obriger Schilderung erfüllt, erstellt.

2.2. Die MySQL Familie

Die MySQL-Famile kann grundsätzlich in zwei Kategorien von zu Testkandidaten geteilt werden. Einzelserver auf der einen Seite sowie Cluster-Lösungen auf der anderen. Dabei wurden auf der Seite der Einzelserver

MySQL, MariaDB und Percona DB betrachtet. Auf der Seite der Cluster-Lösungen kam Galera-Cluster aufgrund von gewissen Vorteilen, welche später näher erörtert werden in die Testauswahl.

2.2.1. MySQL, MariaDB, Percona DB

MySQL ist eines der am weit verbreitetsten [4] relationalen Datenbanksysteme der Welt. Sie gehört zur Gruppe der Open-Source-Datenbanken, verfügt allerdings ebenso über einen kommerziellen Zweig. Dazu wurde als rechtlicher Rahmen eine duale Lizenzierung gewählt, welche einerseits durch die GPL und andererseits durch eine proprietäre Lizenz realisiert wurde [15]. MySQL bildet die Basis für viele Webapplikationen und -systeme. MySQL wird seit 1994 aktiv entwickelt [14] und gehört seit 2010 zum Oracle-Konzern [16].

Die Plattformunterstützung umfasst Windows, einige Linux- und Unix-Varianten sowie macOS. Um eine möglichst gute Performance bieten zu können, ist das System zu großen Teilen in den Sprachen C und C++ implementiert. [14]

Nach dem Kauf von MySQL durch Sun kommt es zu einem Fork des Projektes durch einen der MySQL-Gründer, Ulf Michael Widenius. Dieser hatte Vorbehalte gegen den Kauf und gründete daher MariaDB. Dieses Projekt versteht sich als Drop-In-Replacement von MySQL und wird mittlerweile von vielen anerkannten Projekten, beispielweise Wikipedia, produktiv

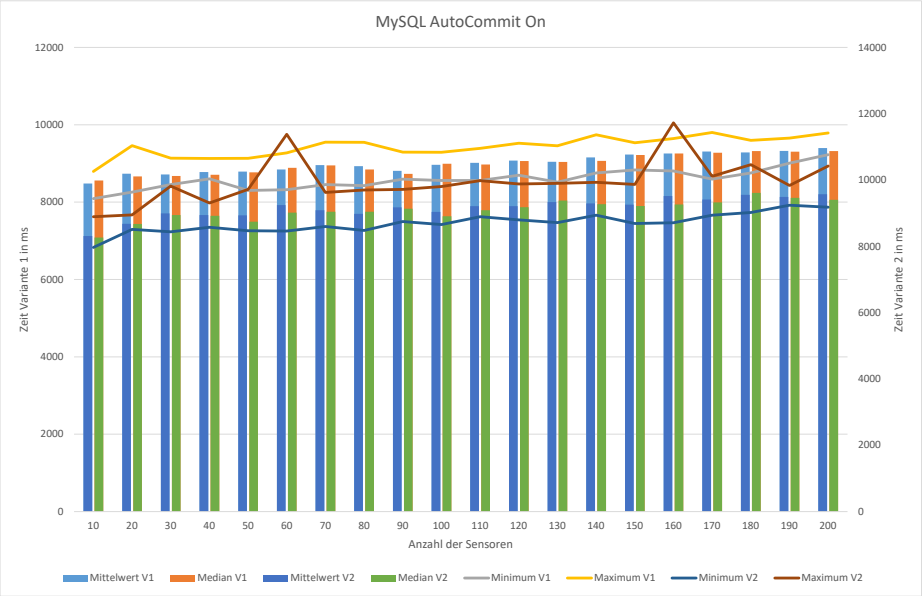

Abbildung 2.1.: MySQL Insertion, V1 und V2, AutoCommit aktiv

eingesetzt. Auch einige Linux-Distributionen wie ArchLinux oder Fedora setzen auf MariaDB als Ersatz für MySQL.

Ergebnisse

Für die Analyse von MySQL wurde die Version 5.7.17 verwendet.

Zu den beiden weiteren Kandidaten MariaDB und Percona DB kann gesagt werden, dass das Verhalten bei der Einfügung gleich zu dem von MySQL

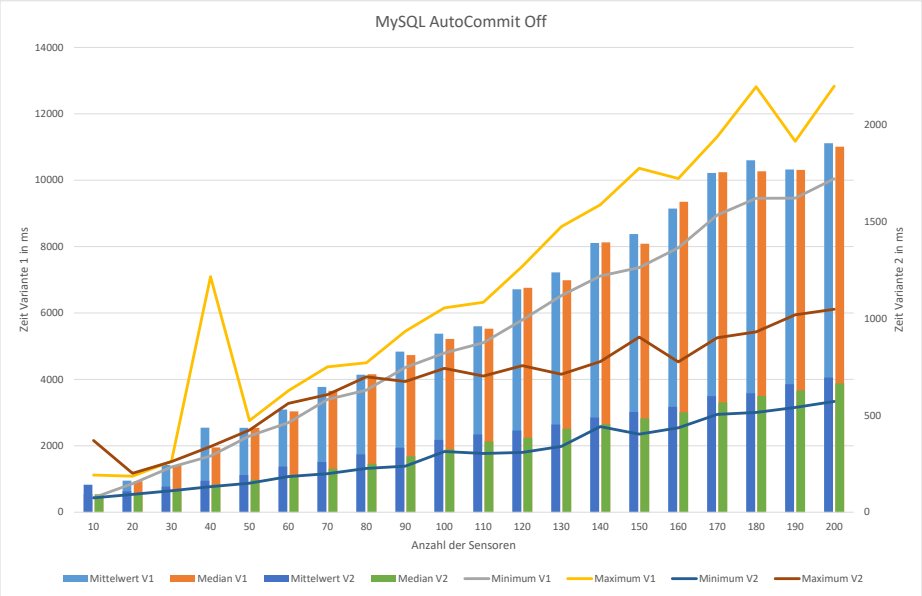

Abbildung 2.2.: MySQL Insertion, V1 und V2, AutoCommit inaktiv

ist. Dadurch kann auf die Darstellung der einzelnen Ergebnisse verzichtet werden.

2.2.2. Galera Cluster

Bei Galera Cluster handelt es sich um einen Replikationscluster der-MySQL Familie. Hierbei steht vorallem Hochverfügbarkeit im Vordergrund. Es setzt auf eine Aktiv/Aktiv-Multi-Master-Topologie und will damit die Probleme der MySQL-Master-Slave-Replikation lösen. Galera sticht im Vergleich zur

	Variante 1 (in ms)	**Variante 2** (in ms)
Query 1	22173	3401
Query 2	253	2326
Query 3	45275	45
Query 4	45	78
Query 5	40639	N/A
Query 6	52612	N/A
Query 7	51	32

Tabelle 2.1.: Abfragen MySQL

Master-Slave-Replikation durch einige Features hervor. Es ermöglicht eine Replikation auf Zeilenebene, was einen wesentlichen Vorteil für die Konsistenz der Daten mit sich bringt. Der Schreibdurchsatz skaliert zwar nur in einem gewissen Rahmen, der Lesezugriff allerdings durch erhöhen der Cluster-Knoten beliebig. [25]

Ergebnisse

Der Test wurde mit Galera Cluster für MariaDB in der Version 10.1 durchgeführt. Aufgrund der Clusterinfrastruktur sind bei Einfügung mit Autocommit höhere Testdauern zu erkennen, wobei das Verhalten grundsätzlich mit dem der Standard-MySQL-Datenbank vergleichbar ist. Es kommt zu einem langsamen Anstieg der Dauern bei einem hohen, in diesem Fall höheren, konstanten Anteil. Zur Veranschaulichung wurde dies in Abbildung 2.3

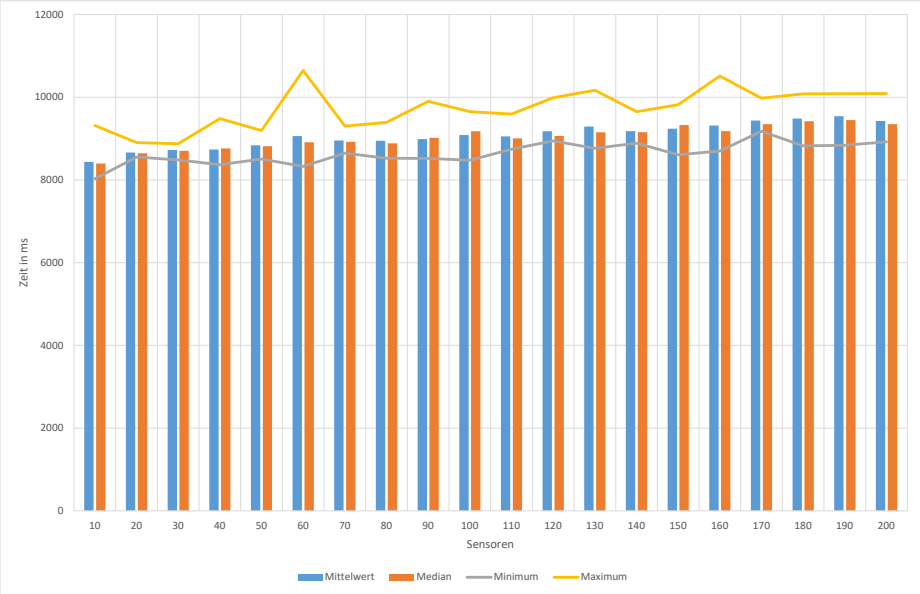

Abbildung 2.3.: Galera Einfügung, Variante 2, AutoCommit aktiv

dargestellt. Bei den Testläufen ohne AutoCommit kommt es ebenfalls zu einem vergleichbaren Verhalten der Datenbank, was in Abbildung 2.4.

Im Bereich der Abfragen, welche in Tabelle 2.2 aufgeführt sind, ist ebenso keine Veränderung des Laufzeitverhaltens zu erkennen, was allerdings dadurch zu erklären ist, dass die Abfrage auf einem Cluster-Node ausgeführt wird. Vorteile aus dem Cluster sollten sich erst bei einer sehr hohen Anzahl paralleler Zugriffe ergeben. Nachteile entstehen bei den Queries durch den Cluster jedenfalls nicht.

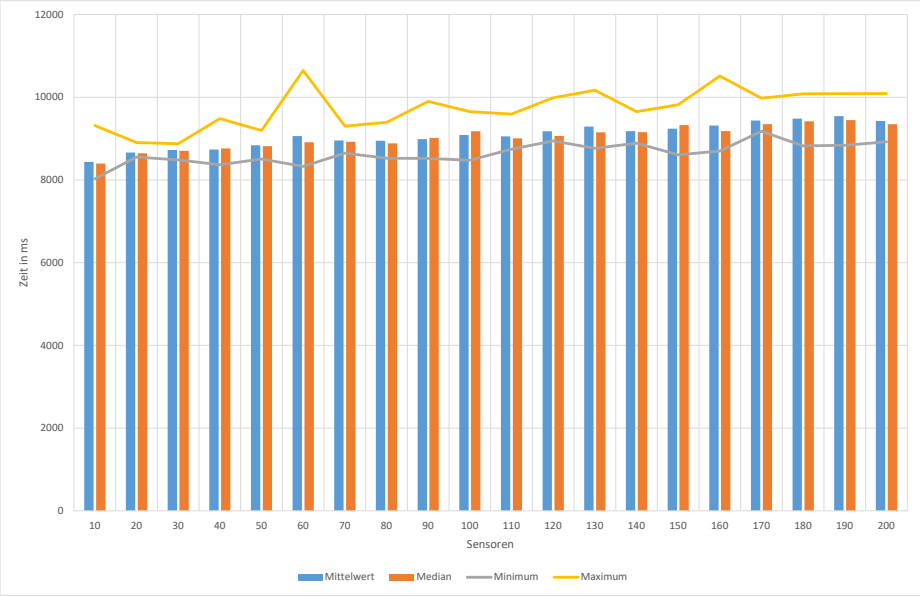

Abbildung 2.4.: Galera Einfügung, Variante 1 und 2, AutoCommit inaktiv

2.3. PostgreSQL

PostgreSQL ist ein freies und quelloffenes RDBMS. Sie gehört wie MySQL zu den weit verbreitetsten Datenbanken der Welt [4]. Die Entwicklung begann an der University of California at Berkeley in den 1980er Jahren und seit 1996 erfolgt sie durch die PostgreSQL Global Development Group [19]. Als rechtliche Grundlage dient dem Projekt die PostgreSQL-Lizenz, welche Ähnlichkeiten zur MIT- oder BSD-Lizenz aufweist [20].

	Variante 1 (in ms)	**Variante 2** (in ms)
Query 1	21985	3211
Query 2	273	3223
Query 3	44225	53
Query 4	65	51
Query 5	39648	N/A
Query 6	55133	N/A
Query 7	41	45

Tabelle 2.2.: Abfragen Galera

Ergebnisse

Für die Analyse wurde PostgreSQL in der Version 9.6.1 verwendet.

Zu Beginn der Auswertungen stehen die Insertion-Operationen. Hierbei zeigt Abbildung 2.5 das Verhalten bei Einzelwerten (Variante 1) mit AutoCommit. Hierbei ist ein klassisches lineares Verhalten mit Anstieg der Sensoren zu erkennen.

In Abbildung 2.7 zeigt die selbe Operation ohne AutoCommits. Hierbei fällt auf, dass das Laufzeitverhalten bei aktivierter Option signifikant schlechter ist als bei deaktivierter Funktion (vgl. Abbildung 2.5). Dies ist dem erhöhten Zugriff auf den Festspeicher geschuldet.

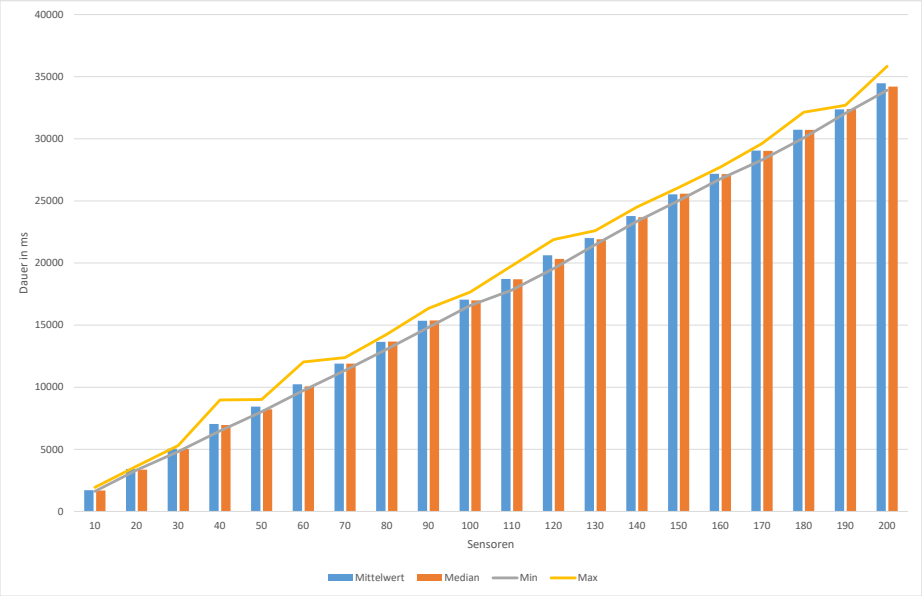

Abbildung 2.5.: PostgreSQL Insertion, Variante 1, AutoCommit aktiviert

2.4. Oracle DB

Oracle DB ist ein kommerzielles Produkt der Oracle Inc., zu welcher auch MySQL gehört. Sie ist die am weitesten verbreitete relationalen Datenbanksystemen weltweit [4]. Sie wird vor allem im Bereich der Großrechner eingesetzt, da sie für diesen Einsatzzweck über Optimierungen verfügt. Die Anzahl der verwendbaren CPU-Kerne und des maximal nutzbaren Hauptspeichers sprechen hier für diese Datenbank.

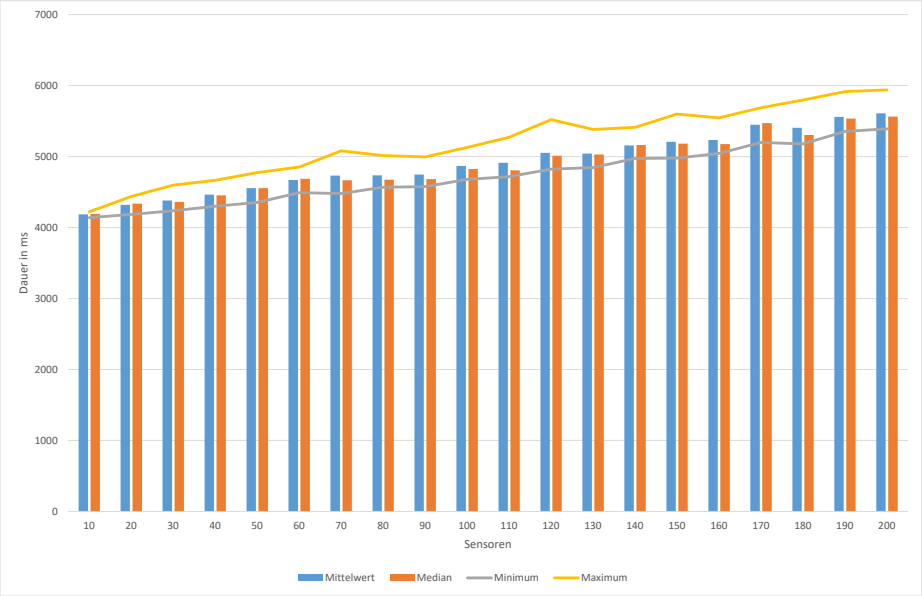

Abbildung 2.6.: PostgreSQL Insertion, Variante 1, AutoCommit aktiviert

Als Testversion wurde 12c in der Standard Edition gewählt, da dies das akutelle Release des Systems ist.

Ergebnisse

Die Insertion-Tests zeigen eine gute Performance. In der Abbildung 2.8 sieht die Insertiongeschwindigkeiten der Varianten 1 und 2. Hierbei fällt auf das

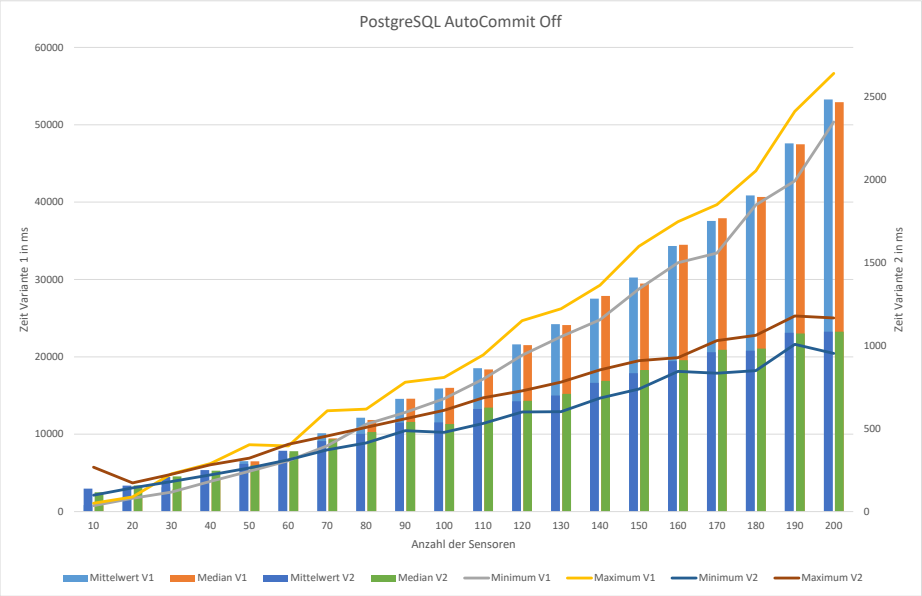

Abbildung 2.7.: PostgreSQL Insertion, Variante 1 und 2, AutoCommit deaktiviert

Ihre Stärken kann die Datenbank allerdings bei den Abfragen ausspielen. Dort zeigt sie eine gute Performance (vgl. Tabelle 2.4).

2.5. IBM DB2

DB2 ist eine kommerzielles RDBMS des Unternehmens IBM Inc. welches seit dem Jahre 1970 entwickelt wird. Es befindet sich unter den Top 10 Datenbanksystemen weltweit [4]. Das, in C entwickelte, RDBMS unterstützt

	Variante 1 (in ms)	**Variante 2** (in ms)
Query 1	12981	930
Query 2	171	85
Query 3	2407	120
Query 4	75	86
Query 5	3109	-
Query 6	114841	-
Query 7	102	60

Tabelle 2.3.: Abfragen PostgreSQL

neben Linux, Unix und Windows (Version LUW) auch das IBM Mainframe Betriebssystem z/OS. Untersucht wurde die Version 10.5.1 LUW.

Der Treiber scheint die AutoCommit-Option zu ignorieren, da dies zu keiner Veränderung des Verhaltens führt. Daher werden bei der Auswertung entsprechend nur diese Grafiken angeführt.

Ergebnisse

Bei der Auswertung der DB2 ergab sich ein ähnliches Bild. In Abbildung 2.9 sieht man sehr klar ein ebenso lineares Einfügungsverhalten, wie bei den meisten anderen Systemen.

Auch bei den Abfragen zeigt sich ein solides Bild was die Performance der einzelnen Abfragen betrifft (vgl. Tabelle 2.5).

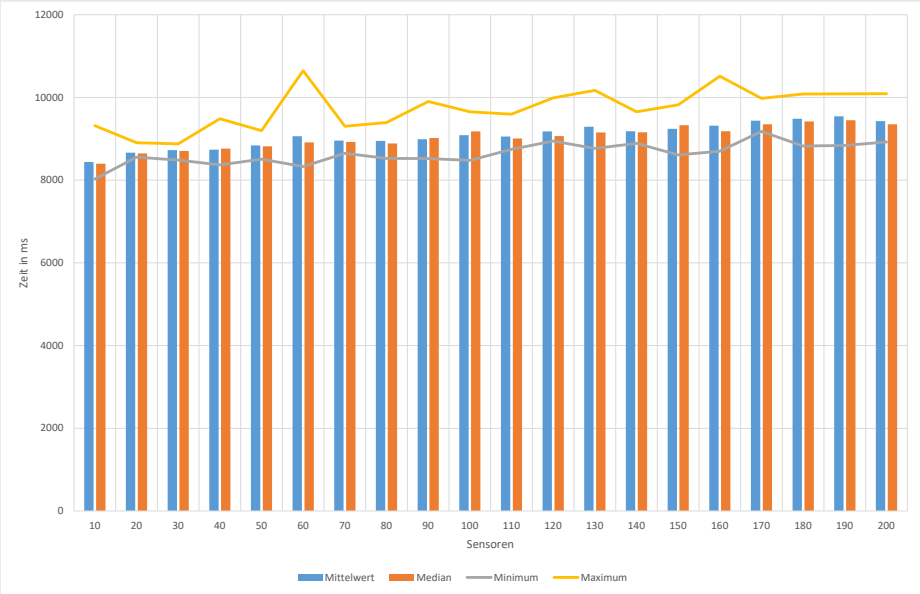

Abbildung 2.8.: Oracle Insertion, V1 und V2, AutoCommit deaktiviert

	Variante 1 (in ms)	**Variante 2** (in ms)
Query 1	1651	807
Query 2	242	134
Query 3	3477	429
Query 4	707	537
Query 5	8072	-
Query 6	61503	-
Query 7	707	422

Tabelle 2.4.: Dauer Abfragen Oracle in ms

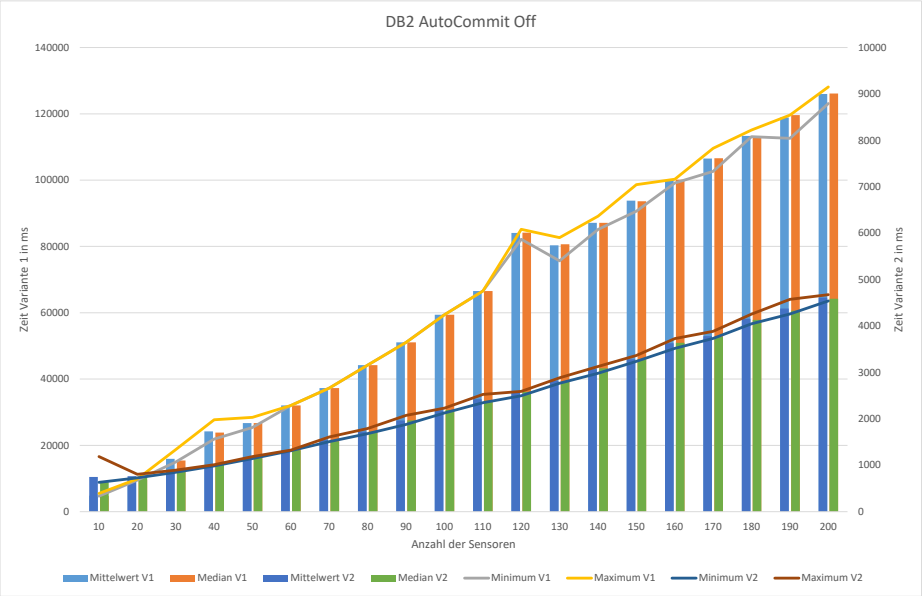

Abbildung 2.9.: DB2 Insertion Variante 1 und 2

	Variante 1 (in ms)	**Variante 2** (in ms)
Query 1	2651	946
Query 2	942	523
Query 3	3477	1459
Query 4	1707	767
Query 5	7072	-
Query 6	71503	-
Query 7	903	324

Tabelle 2.5.: Abfragen DB2

3. Strukturierte Datenspeicher

Dieses Kapitel beschäftigt sich mit den NoSQL-Systemen im Allgemeinen und Speziellen.

Aufgrund der Unterschiedlichkeit der Systeme kann hier keine einheitliche Datenstruktur definiert werden. Auch bei der Implementierung der Einfügung- und Abfrageroutinen muss hier jeweils auf die einzelnen Systeme eingegangen werden. Dadurch wird im Gegensatz zum Kapitel SQL hier die Implementierung für jedes System einzeln betrachtet.

3.1. Grundlagen

Die Klasse der strukturieren Datenspeicher umfasst vor allem Systeme, welche nicht in die Gruppe der klassischen RDBMS fallen.

3.1.1. BASE

Basically Available, Soft state, Eventual Consistency (BASE) ist das Gegenkonzept zu ACID und weicht damit die harten von ACID aufgestellten Regeln für NoSQL auf. Die Abkürzung wurde vorallem wegen ihres Gegensätzlichen Charakters und weniger wegen ihres Inhaltes gewählt. BASE steht grundsätzlich dafür, das Abstriche bei der Konsistenz zu Gunsten der Verfügbarkeit des Systems gemacht werden. Es ist dabei möglich, dass das Gesamtsystem für einen gewissen Zeitraum Inkonsistenzen aufweist. Diese Form der Definition von Konsistenz unterscheidet sich daher von der in ACID.

Es kommt neben grundlegender Designentscheidungen beim NoSQL-Datenspeicher auch oft auf die Konfiguration des Systems an um valide Aussagen über die Konsistenz der Daten nach deren Änderung zu tätigen. Hierbei wird von einigen Systemen ein hybrider Ansatz verfolgt.

3.1.2. Abfragen

Grundsätzlich sind bei den NoSQL-Datenspeichern Abfragen in unterschiedlichster Gestalt möglich. Manche Systeme unterstützen dabei SQL-ähnliche Sprachansätze, andere beschränken sich auf den schlichten Abruf der Daten. Dadurch kann eine Vergleichbarkeit der einzelnen Systeme nur in den Grundfunktionen erreicht werden.

Der erste Fall besteht aus der Abfrage der Anzahl der gespeicherten Datentupel für einen Sensor. Der zweite Testfall umfasst das Abrufen der Daten eines Sensors. Der dritte Fall besteht in einem Abruf der Daten eines definierten Zeitraumes, sofern diese Funktionalität unterstützt wird.

3.2. Dokumentenorientierte Datenbanken

3.2.1. CouchDB

CouchDB (Cluster of unreliable commodity hardware Data Base) gehört als System zur Gruppe der dokumentorientieren Datenbanken. Sie wird seit dem Jahre 2005 als freie Software unter der Apache Lizenz entwickelt und verbreitet. CouchDB wird in großen Teilen in der Programmiersprache Erlang entwickelt.[7]

CouchDB verfolgt einen Offline-First-Ansatz. Dieser ist aufgrund eines Replikationsprotokolls möglich, welches Daten zwischen unterschiedlichen CouchDB-Instanzen austauschen und syncronisieren kann. Dies hat den Vorteil, dass Applikationen nicht in ständigen Kontakt zu einer Online-Datenbank stehen müssen, sofern die Anwendung dies zulässt. Nichtsdestotrotz verfügt CouchDB ebenfalls über die Möglichkeit Datenbanken über Cluster zu verteilen. Als Vorteil ergibt sich hier eine höhere Kapazität und Verfügbarkeit, ohne an der verwendeten API irgendetwas zu ändern. Der

Zugriff erfolgt aufgrund der hohen Kompatibilität über ein HTTP/JSON. Interface.[2]

Grundsätzlich kann eine CouchDB-Instanz auf jedem herkömmlichen Device verwendet werden, was auch durch Langform des Namens ausgedrückt wird. Dabei werden auch Mobile Devices inkludiert. [7]

Implementierung

Das System selbst wurde in der Version 2.0 eingesetzt. Der Test-Client wurde mit der Java-Client-Library ektorp erstellt, welche eine ausgereifte Implementierung darstellt. Es werden hierbei POJOs verwendet. Diese werden in Dokumente umgewandelt und in die CouchDB mit einem Key eingefügt. Die Abfrage funktioniert umgekehrt genauso. Daten werden von CouchDB gelesen und in POJOs verpackt und zur Verfügung gestellt.

Es wurde hier die Fälle 50, 100, 150 und 200 Sensoren abgedeckt. Die POJOs enthalten hier jeweils die gewünschte Anzahl an Sensoren. Hierbei ist erkennbar, dass die Einfügungszeiten (vgl. Abbildung 3.1) konstant sind. Es kommt dabei offensichtlich auf die Anzahl der Dokumente und nicht auf die Größe derer an. Einzeldokumente pro Wert wurden als unpraktikabel und viel zu zeitintensiv klassifiziert und daher nicht weiter verfolgt.

Im Code Sample 3.1 wird ein Dokument erstellt und in den Datenspeicher eingefügt. Das einzufügende Objekt beinhaltet die Daten von 200 Sensoren

als Member-Variablen die in diesem Fall im Konstruktor per Zufall erstellt werden.

```
CouchDbInstance dbInstance = new StdCouchDbInstance(
    httpClient);
CouchDbConnector db = new StdCouchDbConnector("testdb",
    dbInstance);

for (int it = 0; it < 10; it++) {
        long start = System.currentTimeMillis();
        for (int i = 0; i < 1_000; i++) {
            db.create(key, new MeasDataAll(System.
                currentTimeMillis(), "gen1"));
        }
        System.out.println(System.currentTimeMillis() -
            start);
}
```

Code Sample 3.1: Einfügung in CouchDB

Abfragefunktionen können in JavaScript implementiert werden um spezielle Bedingungen oder Ähnliches abzufragen. Diese Funktionaliät wurde allerdings in der aktuellen Implementierung außen vor gelassen, da die gewünschten Abfragen auch ohne dies möglich waren.

	Mittelwert	Median	Minimum	Maximum
50	20097	20122	19691	20219
200	20608	20584	20336	20934

Tabelle 3.1.: Auswertung der Einfügungstests für CouchDB (in ms)

Ergebnisse

Die Auswertung des Einfügungstests zeigt, dass CouchDB ein, wie in Tabelle 3.1 von der Größe des einzufügenden Dokumentes unabhängiges Verhalten zeigt.

Die Abfragedauern zeigen im Gegensatz zu den Einfügungszeiten eine lineare Komponente.

3.2.2. MongoDB

MongoDB, abgekürzt für humongous database, ist ebenfalls ein Vertreter der dokumentenorientierten NoSQL-Datenspeicher. Begonnen wurde mit der Entwicklung im Jahr 2007, die Veröffentlichung fand 2009 statt [23]. Sie wird von der Firma MongoDB Inc. betreibt das Project unter der GNU AGPL v3 sowie der Apache-Lizenz, damit ist MongoDB Open Source. Derzeit ist MongoDB vor Cassandra die verbreitetste NoSQL-Datenspeicher [4].

MongoDB speichert die Daten in einem binären Dokumentformat basierend auf JSON [10]. Dadurch wird eine höhere Effizienz im Bereich Durchsuch-

barkeit und Speicherplatzbedarf erreicht [3]. Für die Analyse wurde die Version 3.4.1 verwendet.

Implementierung

Bei der Implementierung des Clients wurden zwei verschiedene Varianten implementiert, welche an die der Clientimplementierung der SQL-Systeme angelehnt ist. Hierbei wurden zwei unterschiedliche Datenstrukturen verwendet.

In der ersten Struktur (siehe Code Sample 3.2) werden die Werte einzeln in separaten Dokumenten gespeichert. In der zweiten Variante (siehe Code Sample 3.3) werden die Daten aller Sensoren in ein gemeinsames Dokument gespeichert. Zum Unterschied zu Variante 1, bei welcher n Dokumente entstehen pro Zeitpunkt entstehen, wird bei Variante 2 nur ein einzelnes Dokument erstellt.

```
{
        "Facility": "anlage" + integer,
        "Sensor": "senor" + integer,
        "Value": decimal,
        "Time": integer
}
```

Code Sample 3.2: Variante 1 der MongoDB-Struktur

```
{
        "Facility": "anlage" + integer,
        "sensor1": decimal,
        "sensor2": decimal,
        ....
        "sensor(n)": decimal,
        "Time": integer
}
```

Code Sample 3.3: Variante 2 der MongoDB-Struktur

Es werden abermals 1000 Zeitpunkte zu je zehn bis 200 Sensoren in die Datenbank eingefügt. Für den Abfragetest werden 100 Anlagen mit je 100 Sensoren zu 10000 Zeitpunkten eingefügt jeweils für Variante 1 und Variante 2.

MongoDB unterstützt den Einsatz von Schlüsseln, was eine effizientere Ausführung von Abfragen ermöglicht. Ohne die Schlüssel muss das System eine Scan über alle Dokumente in einer Collection durchführen um die gewünschten zu selektieren. Es können neben dem primären Schlüssel auch sekundäre zur Performancesteigerung definiert werden. [9]

Ergebnisse

Der Test-Client für die Einfügung erzielte eine sehr gute Performance (vgl. Abbildung 3.1) obwohl die Art der Daten nicht optimal für eine dokumentorientierten Datenspeicher sind.

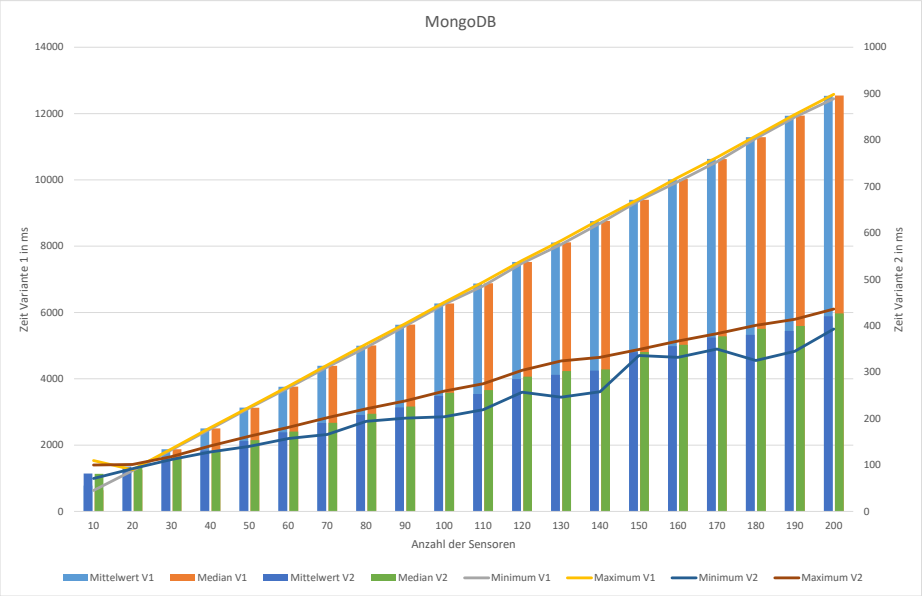

Abbildung 3.1.: MongoDB Einfügung, Variante 1 und 2

Bei den Abfragen zeigt sich ein geteiltes Bild. Hierbei hängt es davon ab ob Sekundärschlüssel zum Einsatz kommen oder nicht. Bei dessen Verwendung steigt die Performance signifikant, was der Tabelle 3.2 entnommen werden kann.

	Variante 1	Variante 2
Query 1	51	6
Query 2	90142	14981
Query 3	59871	3811
Query 1 mit Schlüssel	9	3
Query 2 mit Schlüssel	53101	5598
Query 3 mit Schlüssel	46	7

Tabelle 3.2.: Auswertung der Abfrage für MongoDB (in ms)

3.3. Spaltenorientierte Datenbanken

3.3.1. Apache Cassandra

Apache Cassandra ist ein spaltenorientierter Datenspeicher. Er besitzt eine hohe Fehlertoleranz aufgrund seiner verteilten Architektur. Hierbei erfolgt eine Verteilung und Replikation der Daten auf mehrere Nodes, was eine hoher Performance und Ausfallssicherheit unterstützt. Das System kann flexibel an die verschiedenen Anforderungen von unterschiedlichen Anwendungen angepasst und optimiert werden. Es hat besonders Stärken im Bereich Skalierbarkeit. Als Referenz gibt das Projekt unter anderem eine Anwendung von Apple, welche 75.000 Nodes umfassen soll,in welchen über 10 Petabyte an Daten gespeichert sein sollen, an. [1]

Das Projekt Cassandra verfügt über einige kommerzielle Angebote, welche

Support für Enterprise-Kunden bieten. Hierbei sind vorallem DataStax und Cloudera zu erwähnen.

Ein Cassandra-Cluster ist von seiner Topologie als Ring angeordnet. Das System verteilt die Daten über standardmäßig die Daten über den gesamten Cluster. Dies erfolgt nach einer randomisierten Strategie und ermöglicht somit eine gleichmäßige Verteilung über den Cluster. Dabei kommt weiters eine Replikationsstrategie zum Einsatz um die Daten ausfallsicher zu speichern. Hierbei muss definiert werden wie oft die Daten im Cluster vorhanden sein sollen. Dabei kann auch auf eine Verteilung über mehrere physische Datencenter Rücksicht genommen werden.

Für die Analyse wurde die Version 3.0.10 verwendet.

Implementierung

Bei der Implementierung des Cassandra-Clients wurde eine Struktur gewählt, welche auf eine gute Verteilbarkeit auf den einzelnen Nodes begünstigt. Dies wird dadurch erreicht, indem in den Key, welcher grundsätzlich aus Anlage- und Sensor-Id besteht, eine zeitliche Komponente und somit eine Teilung aufgenommen wird. Der Key gestaltet sich daher wie folgt: *aid-sid-date-h* wobei *aid* der Anlagenidentifier, *sid* der Sensorenidentifier, *date* das akutelle Kalenderdatum und *h* die Stunde der Aufzeichnung darstellt. Dadurch werden Datenblöcke erzeugt, welche einfach über einen Cluster verteilt werden können. Bei einer Variante ohne Zeitkomponente kommt es

zu Problemen bei der Abfragbarkeit der Daten, aufgrund der wachsenden Größe.

Als Einfüge- und Abfragesprache dient Cassandra die Sprache CQL. Diese kann als Subset von SQL angesehen werden. Die Sprache erlaubt nur eingeschränkt Filterungen an den Daten vorzunehmen.

Ergebnisse

Der Einfügungstest wurde mit einem Node in der Standardkonfiguration durchgeführt. Die dabei erhaltenen Daten decken sich mit ähnlichen Versuchen (vgl. [21]) bei welchen allerdings eine sehr starke Skalierbarkeit des Systems erkennbar ist, da auch mit mehreren Nodes gemessen werden konnte. Dies ist aufgrund der Testkonfiguration eingeschränkt.

Die Abfragen konnten eine ausgezeichnete Performance an den Tag legen. Die erste Query wurde vom System in 81 ms erledigt, die zweite in 37 ms und die dritte in 17 ms. Damit zeigt Cassandra klar ihre Stärke in der raschen Abfrage von Daten.

3.3.2. InfluxDB

InfluxDB ist ein Open-Source-Zeitreihendatenbank entwickelt von InfluxData unter der MIT-Lizenz. Das System wurde und wird aus Performancegründen in der Sprache Go entwickelt. Optimiert als Hochverfügbar-

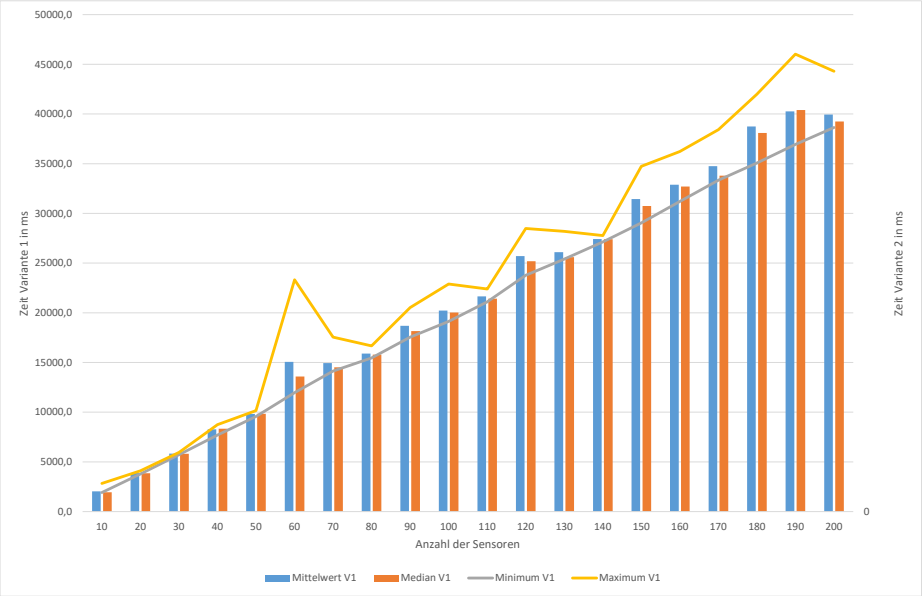

Abbildung 3.2.: Cassandra Einfügung, 1 Node

keitsspeichersystem speichert es Zeitreihendaten wie Speicherauslastung, Temperaturen und viele andere Messungen. Es steht eine HTTP-API zur Verfügung, welche auf einfache Weise Speicherung und Zugriff der Daten erlaubt.

Das Projekt wird ebenso kommerziell unter dem Namen Influx Enterprise vermarktet. Hierbei handelt es sich um eine Cloud-Lösung, welche um weitere Funktionalitäten erweitert wurde. Für die Analyse wurde die Version 1.1.0 verwendet.

Implementierung

InfluxDB verfügt sehr unterschiedliche Möglichkeiten Daten einzufügen. Im Client wurde die HTTP-JSON-API als Einfügungsmethode implementiert. Das System verfügt über eine SQL-ähnliche Einfügungs- und Abfragesprache, was den Einstieg für RDBMS-Umsteiger erleichtern soll. Als Datenstruktur wurde eine Multikanalstruktur gewählt. Diese zeigte bei allen Systemen einen

Ergebnisse

Bei den Abfragen zeigte sich bei InfluxDB eine hervorragende Performance. Die erste Abfrage wurde in 25 ms erledigt, bei der zweiten Abfragen vergingen 262 ms bis zum Abschluss und die dritte war in 162 ms erledigt.

3.4. Key-Value-Caches und - Stores

3.4.1. Redis

Redis ist mit Abstand der verbreitetste Key-Value-Stores und gehört zur Gruppe der datenstrukturorientieren In-Memory Datenbanken. Aufgrund der verwendeten Schüssel-Wert-Datenstrukturen gehört sie zur Familie der NoSQL-Datenbanken. Einer der großen Vorteile besteht gerade in dieser

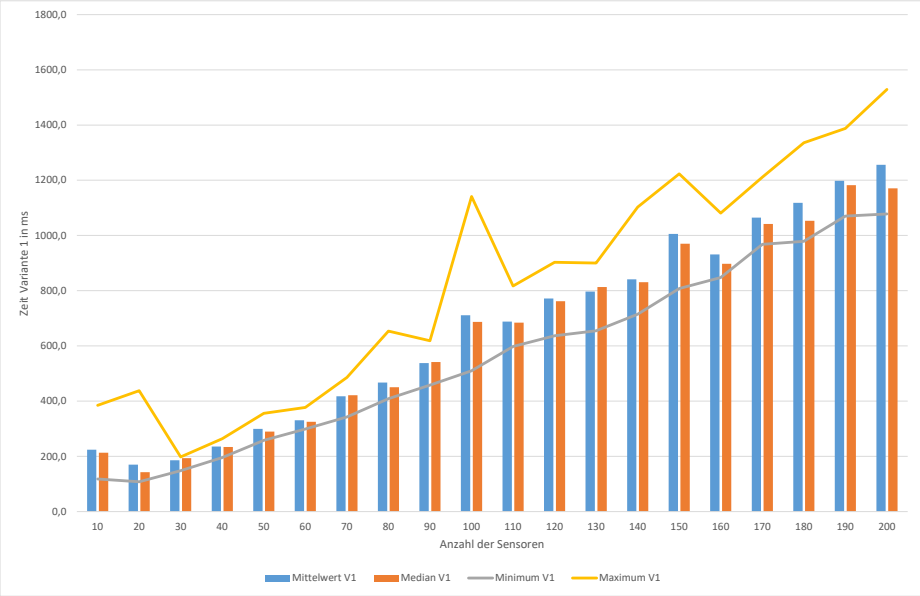

Abbildung 3.3.: InfluxDB Einfügung

einfachen Struktur, welche zwar nicht unbedingt für komplexe Datenstrukturen geeignet ist, allerdings einen enormen Geschwindigkeitsvorteil bietet. Grundsätzlich verfügt das Projekt über verschiedene Möglichkeiten Daten zu speichern, von der einfachen Liste über Sets und HashMaps bis hin zu Geolocation-behafteten Daten.

Das System ist in ANSI C implementiert und kann auf vielen unixoiden Systemen problemlos ausgeführt werden. Ein Windows-Support wird derzeit nicht offiziell angeboten. Als rechtlicher Rahmen fungiert die BSD Lizenz. Das Projekt verfügt einerseits über eine große und aktive Community und

andererseits wird auch kommerzieller Support für Firmen angeboten. Für die Analyse wurde die Version 3.0.6 verwendet.

Redis unterstützt eine Master-Slave-Replikation und kann seit Version 3.0 auch als Cluster betrieben werden. Bei der Verwendung als Cluster werden die Daten über Partitionierung der Schlüssel auf die einzelnen Nodes aufgeteilt. Sofern der Cluster als Cache fungiert, ist eine randomisierte Verteilung empfohlen. Bei längerfristiger Datenspeicherung wird davon abgeraten. Beim Hinzufügen neuer Nodes muss ein manuelles Rebalancing, eine Neuaufteilung der Daten, durchgeführt werden. [18]

Implementierung

Redis wird über ein Set von Kommandos gesteuert, in unserem Fall kommen ZADD, ZRANGE, ZCOUNT zum Einsatz. In Code-Sample 3.4.1 sieht man wie die Daten in das System eingefügt werden.

```
for (int ins = 1; ins <= sensors; ins += 10) {
        System.out.print(ins);
        for (int run = 0; run < maxRuns; run++) {
                lStartTime = System.currentTimeMillis();
                for (int i = 0; i < insertions; i++) {
                        for (int v = 0; v < ins; v++) {
                                random = Math.random() * 100 + 1;
                                cmd.zadd(measId+v, System.
                                    currentTimeMillis(), String.
```

```
                                format(Locale.ENGLISH, "%.4f",
                                random));
                        }
                    }
                    lEndTime = System.currentTimeMillis();
                    System.out.print(";" + (lEndTime - lStartTime));
                    for (int v = 0; v < ins; v++)
                        clean.del(measId + v);
                }
                System.out.println("");
            }
```

Bei Code-Sample 3.4 sind die Abfragen in Form von integierten Funktionen angeführt.

```
List<String> zrange = cmd.zrange(measId, 0, -1);
//bzw.
long elems = cmd.zcount(measId, "-inf", "+inf");
```

Code Sample 3.4: Redis - Abfragen

Bei der Client-Implementierung für Redis wurden abermals 2 Varianten umgesetzt. Die Variante 1 speichert jeden Sensor in einer eigenen Datenstruktur. Bei Variante 2 werden die Daten Strichpunkt getrennt gespeichert. Dies führt dazu, dass die Anzahl der Einfügungen pro Zeitschritt von Variante 1 n und von Variante 2 eins beträgt.

Ergebnisse

Bei der Auswertung der zwei Varianten zeigt sich ein bekanntes Bild. Die Einfügungen der Variante 1 benötigen klarerweise länger als die zweiten Variante, was mit den geringen Datentuples erklärbar ist. Dahingegen muss hier bei der Abfrage der Daten einen gewissen Nachteil was die Rückgewinnung der Daten betrifft in Kauf nehmen, da man die in Variante 2 komma-separierten Daten erst komplett aufbereiten muss bevor man diese einsetzen kann.

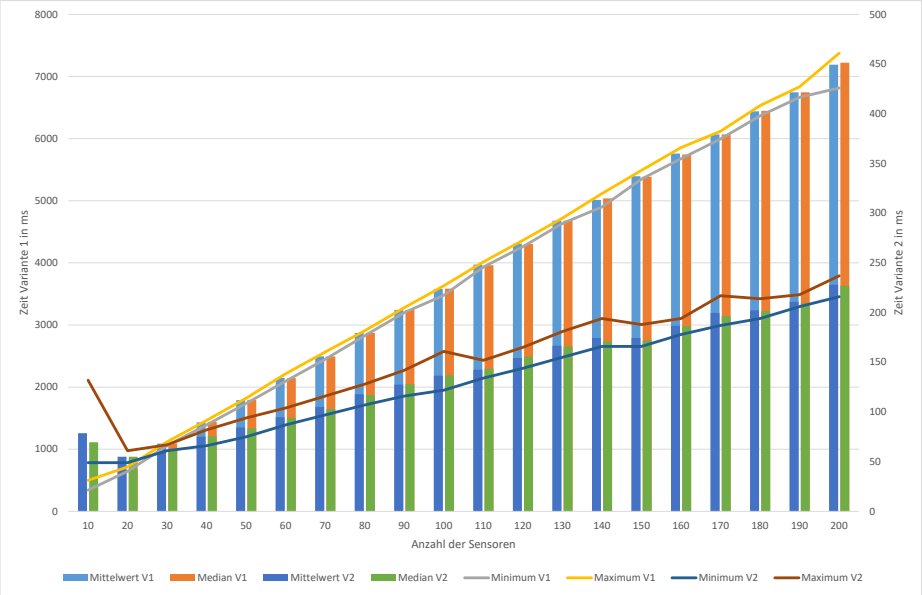

Abbildung 3.4.: Redis Einfügung, Variante 1 und 2

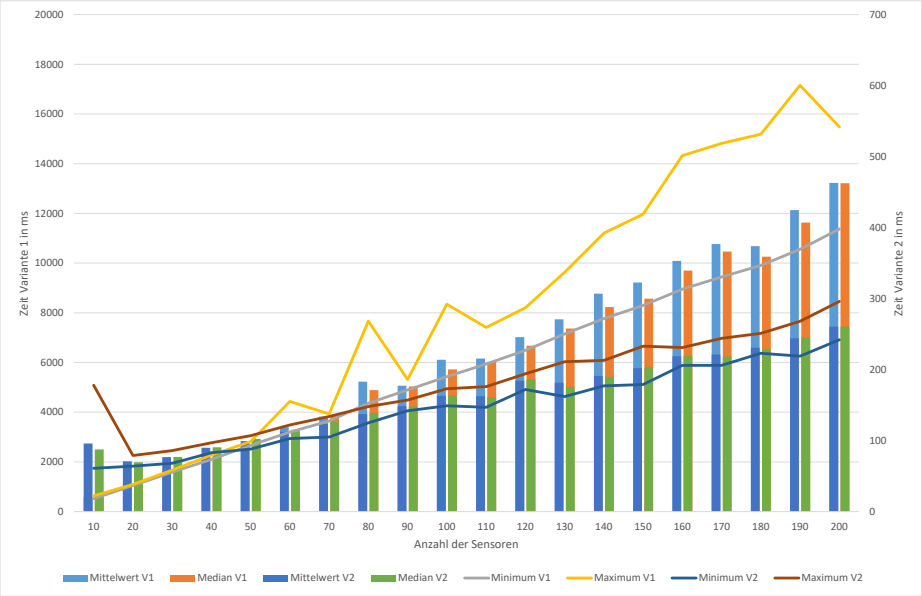

Abbildung 3.5.: Redis 3-Node Cluster Einfügung, Variante 1 und 2

3.4.2. Hazelcast

Hazelcast ist dual-lizenziertes Projekt der Firma Hazelcast, Inc. Der Open-Source-Teil der Software lizenziert unter der Apache Lizenz, kommerziell wird das Projekt unter einer kommerizellen Vereinbarung vermarktet.

Dieses Projekt versteht sich unter anderem als Caching-Lösung, weshalb ein sehr transparenter Zugriff über den Sourcecode auf die einzelnen Speicherstrukturen, wie Listen und Sets, möglich ist.

Implementierung

Bei Hazelcast steht eine umfassende Sammlung an Client-Libraries zur Verfügung. Diese können sehr einfach in ein bestehendes Projekt eingebunden werden. Hierbei wird eine Datenstruktur beim Cluster erstellt und kann danach sofort verwendet werden (vgl. Code Sample 3.5). Aufgrund der unterschiedlichen Strukturen empfiehlt es sich eine MultiMap einzusetzen, da dort eine Liste der Datentuple hinter jeden Key hinterlegt werden kann.

```
MultiMap<String, DataTuple> sen = instance.getMultiMap("sensor01")
    ;
sen.put("data"+(new Date()).toString(), new DataTuple(System.
    currentTimeMillis(), Math.random() * 100));

//bzw.

Map<Long, Double> map = inst.getMap("sensor10");
map.put(System.currentTimeMillis(), Math.random() * 100);
}
```

Code Sample 3.5: Erstellung einer Datenstruktur und Einfügung in Hazelcast

Auch ein Abruf der Daten ist somit mit wenigen Zeilen Code (vgl. Code Sample 3.6) umzusetzen.

Sensoren	Mittelwert	Median	Minimum	Maximum
50	2470,2	2375	2282	3163
100	4714,6	4669	4606	5076
150	7189,1	705	6963	7671
200	9500	9422,5	9296	10339

Tabelle 3.3.: Einfügungen in Hazelcast (in ms)

```
Collection<DataTuple> data = map.getMultiMap("sensor01").get(
    datakey);
```

Code Sample 3.6: Abfrage einer Datenstruktur in Hazelcast

Ergebnisse

Bei Hazelcast wurden die Einfügevorgänge wie bei den anderen Systeme mit verschiedenen Anzahlen von Sensoren getestet. In Tabelle 3.4.2 zeigt sich ein lineares Verhalten bei der Einfügung der Daten. Die Werte stehen für die Ausführungszeit von 1000 Einfügungsvorgänge mit der jeweiligen Anzahl der Sensoren.

Der Zugriff auf die entsprechenden Daten erfolgt in 57 ms für einen Sensor mit 20000 gespeicherten Datenpunkten. Die Daten werden in einer entsprechenden *Collection*<> bereitgestellt. Somit ist ein Zugriff auf die Anzahl ohne Probleme programmatisch möglich.

3.5. Zeitreihendatenbanken

3.5.1. OpenTSDB

OpenTSDB fungiert als HBase-Aufsatz und speichert damit seine Daten in einem Hadoop-Datensystem. Sie wird sowohl unter der LGPL v2.1+ als auch unter der GPL v3+ entwickelt und ist damit ebenfalls Open Source nachdem StumbleUpon das Projekt freigegeben hat.

Sie versteht sich selbst als skalierbare Zeitreihendatenbank, die mit sehr hohen Datenmengen umzugehen vermag. Da, wie bereits oben erwähnt, OpenTSDB als Aufsatz zu HBase und damit Hadoop fungiert, ist Skalierbarkeit kein Problem. Es sind, bei entsprechender Ausstattung des Clusters, Millionen Lese- und Schreibzugriffe pro Sekunde möglich. Auch Kapazitätserweiterungen können durch eine einfache Ergänzung neuer Nodes erreicht werden.

Die Daten werden roh und unreduziert gespeichert. Ebenfalls können sie mit Tags versehen werden, was für die spätere Auswertung erleichtern kann und soll.

Das System stellt die Daten in einfachen Graphen über die integierte WebUI dar. Daten können allerdings auch über die JSON-API abgerufen werden.

Implementierung

OpenTSDB verfügt über ein Telnet-ähnliches-Protokoll sowie über eine HTTP-JSON-API, welche sowohl für die Einfügungen als auch für die Abfragen Verwendung findet. Die Implementierung des Clients erfolgte daher mit der Apache HttpClient-Library in Verbindung mit der Gson-Library von Google. Hierbei wurden die entsprechenden Requests erstellt und übermittelt.

Es werden wie bereits zuvor Zufallsdaten im System gespeichert und danach abgerufen. Die Abfragen können mit verschiedenen Aggregatoren erstellt werden. Hierbei werden die Daten mehrerer Sensor summiert, gemittelt, das Maximum oder Minimum bestimmt oder roh übermittelt. Jede Abfrage muss einen solchen Aggregator sowie einen Zeitbereich angeben. Die Daten werden danach im JSON-Format ausgeliefert.

Ergebnisse

OpenTSDB wurde in der Version 2.2.0 getestet. Hierbei wurde zuerst Daten für bis zu 200 Sensoren eingefügt. Die Dauern der Einfügungen sind in der Abbildung 3.6 ersichtlich. Hierbei ist zu erkennen, dass die Speicherung sehr schnell erfolgt.

Bei den Abfragen kommt es zu einer Laufzeit von 118 ms bei Query 1, von 1675 ms bei Query 2 und zu 1578 ms bei Query 3.

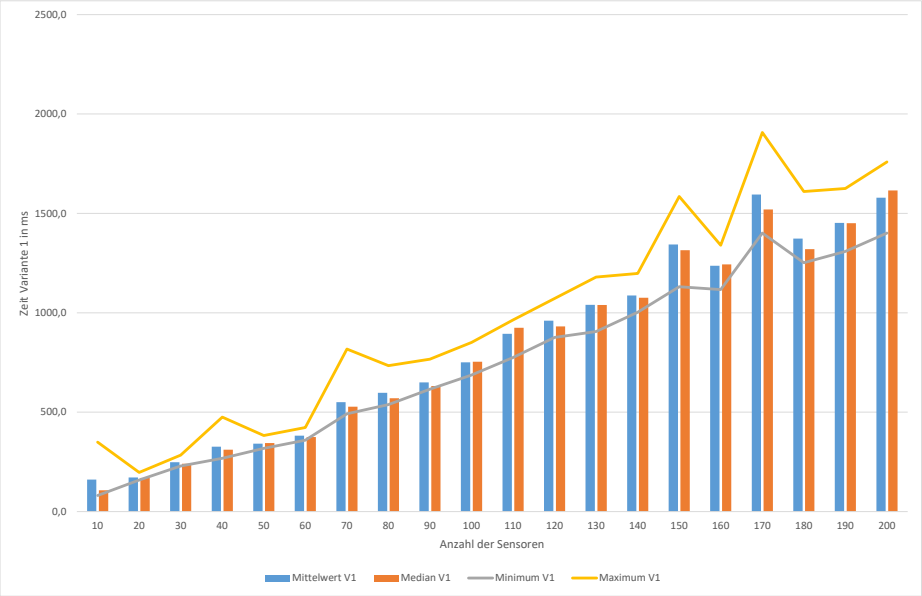

Abbildung 3.6.: Einfügungen OpenTDSB

3.5.2. Riak TS

Riak TS eine weitere Vertreterin der NoSQL-Zeitreihendatenbanken. Sie wird primär von der Firma Basho Technologies Inc. als Open Source Projekt unter der Apache Lizenz entwickelt. Weiters wird auch Support unter einer kommerziellen Lizenz angeboten. Dieses System verfügt ebenso wie InfluxDB über die Möglichkeit die Daten entsprechend zu taggen.

Das System verfügt über eine masterlose Architektur und deren Cluster sind ebenso wie die von Cassandra als Ring angeordnet.[22]

Implementierung

Grundsätzlich verfügt Riak TS über Client-Libraries für alle wichtigen Programmiersprachen. Als Schnittstellensprache fungiert grundsätzlich SQL, wenn auch umfänglich etwas eingeschränkt. Jede Abfrage muss eine WHERE-Bedingung enthalten in welcher alle Schlüsselfelder verwendet werden müssen. Dadurch sind nur Range-Abfrage möglich, was die Anwendung von Query 2 ausschließt.

Im Erstellungsstatement der Tabelle, in Code-Sample 3.7, sieht man klar welche Struktur die Daten aufweisen.

```
CREATE TABLE test (
time timestamp not null,
sensor varchar not null,
value double not null,
Primary key ((sensor, quantum(time, 1, 'h')), sensor, time))";
```

Code Sample 3.7: RiakTS - Tabellenerstellung

Die Einfügungen und Abfragen werden über die Client-API im System implementiert.

Ergebnisse

Bei der Auswertung der Ergebnisse von Riak TS in Tabelle 3.4 zeigt sich abermals klar, dass die Reduktion der Einfügungen zu einer klaren Steigerung der Performance führt.

	Mittelwert	Median	Minimum	Maximum
Variante 1	20207,6	19972,5	19464	22128
Variante 2	726,3	715	613	858

Tabelle 3.4.: Einfügungstests für Riak TS, Zeiten in ms

Bei den Abfragen kommt es bei der ersten zu einer Laufzeit von 53 ms und bei der dritten von 254 ms.

4. Evaluierung und Vergleich

Die Ergebnisse, welche im vorherigen Abschnitt einzeln beleuchtet wurden, müssen nun noch in einem vergleichenden Kontext betrachtet werden. Die Vergleiche finden einerseits innerhalb der jeweiligen Hauptgruppe, SQL und NoSQL, und andererseits im Gesamtkontext statt.

4.1. SQL-Auswertung

Beim Vergleich der einzelnen SQL-Datenbanken kommt es zu starken Unterscheidungen bei der Performance.

Beim Insertion-Teil kommt es zu teils sehr unterschiedlichen Ergebnissen bei den einzelnen Testkandidaten. Wie man in Tabelle 4.1 erkennen kann, tut sich MySQL durch eine sehr schnelle Einfügungsperformance hervor, gefolgt von PostgreSQL. Oracle kann im Bereich der Variante 2 ebenfalls eine gute Performance zeigen.

Sensoren	MySQL		Galera		PostgreSQL		Oracle		DB2	
	V 1	**V 2**	**V 1**	**V 2**	**V 1**	**V 2**	**V 1**	**V 2**	**V 1**	**V 2**
10	546,7	93,6	1547,1	354,2	833,7	138,4	5223,7	644	4824,1	749,4
20	952,0	108,8	1564,6	377,1	1823	156,6	10091,9	845	9614,0	762,8
30	1422,8	131,5	2343,5	388,2	3150,1	209,0	16425,3	1449,5	15954,0	868,4
40	2548,2	162,1	2961,1	401,3	4432,8	250,3	21388,5	831	24240,0	1006,0
50	2563,2	191,9	3693,9	494,4	6508,4	287,9	25567,5	988,5	26720,8	1164,6
60	3093,2	235,5	4385,1	660,1	7507,5	366,6	31222,6	1264	32047,0	1321,8
70	3775,1	259,6	5302,1	741,2	10127,2	424,9	39946,9	1380	37279,0	1543,4
80	4132,1	298,2	5950,1	846,2	12142,3	468,9	49225,6	1506,5	44183,0	1722,4
90	4837,6	333,1	6724,5	873,2	14570,3	536,6	49426,4	1582,5	51089,0	1965,5
100	5377,0	373,6	7317,2	877,2	15919,5	538,5	56490,2	1697,5	59381,0	2178
150	8376,2	517,8	11580,5	952,1	30256,7	834,6	81904,3	2030	93827,1	3284,8
200	11114,4	543,8	15358,1	1088,2	53267,4	1085,4	109872,5	2432	126003,2	4609,6

Tabelle 4.1.: Mittelwert der Einfügedauern der RDBMS (in ms)

	MySQL		Galera		PostgreSQL		Oracle		DB2	
	V 1	**V 2**	**V 1**	**V 2**	**V 1**	**V 2**	**V 1**	**V 2**	**V 1**	**V 2**
Query 1	22173	3401	21985	3211	12981	930	1651	807	2651	946
Query 2	253	2326	273	3223	171	85	242	134	942	523
Query 3	45275	45	44225	53	2407	120	3477	429	3477	1459
Query 4	45	78	65	51	75	86	707	537	1707	767
Query 5	40639	-	39648	-	3109	-	8072	-	7072	-
Query 6	52612	-	55133	-	114814	-	61503	-	71503	-
Query 7	51	32	41	45	102	60	707	422	903	324

Tabelle 4.2.: Abfragedauern aller RDBMS (in ms)

Bei den Abfragen zeichnet sich ebenfalls ein interessantes Bild. Während MySQL bei dein Einfügungen in Führung lag, sind die Abfragen PostgreSQL und Oracle hier meist schneller. In Tabelle 4.2 sieht man klar, dass hier die Geschwindigkeitsvorteile von PostgreSQL und Oracle. Einzig bei der Query 6 kann MySQL deutlich davonziehen. Aufgrund des hohen Speicherverbrauchs von PostgreSQL während der Query liegt die Vermutung Nahe, dass dies mit einer nachträglichen temporären Indizierung der Daten zu tun hat.

4.2. NoSQL-Auswertung

Die vergleichende Auswertung der NoSQL-Kandidaten führte zu den erwarteten Ergebnissen. Die Einfügungen erreichen meist eine sehr hohe Geschwindigkeit. Dies geht meist auf Kosten der Konsistenz und/oder der Ausfallsicherheit Bei den Abfragen kann nur bedingt ein Vergleich durchgeführt werden, da dieser selten auf eine ganz exakt gleiche Basis gestellt werden kann. Es kommt hier sehr stark auf den Funktionsumfang der einzelnen Systeme an.

In Tabelle 4.3 sieht man klar, dass hier Systeme mit spezieller Optimierung, wie Zeitreihendatenbanken oder mit schwacher Strukturierung wie Key-Value-Stores im Vorteil sind.

CouchDB	MongoDB		Cassandra	InfluxDB	Redis		Riak TS	Hazelcast	OpenTSDB
V1&V2	V1	V2			V1	V2			
19871	12532,7	420,3	40120	1378,3	13224,4	228,1	1421	9500	1088,2

Tabelle 4.3.: Einfügedauern (Mittelwert) aller NoSQL-Systeme für 200 Sensoren (in ms)

	CouchDB	MongoDB		Cassandra	InfluxDB	Redis		Riak TS	Hazelcast	OpenTSDB
Abfragen	V1 & V2	V1	V2	V1	V2	V1	V2	V1	V2	V2
Query 1	1971	9	3	81	25	1	2	53	57	118
Query 2	1982	53101	5598	37	262	16	95	NA	57	1675
Query 3	1971	46	7	17	162	12	68	248	NA	1578

Tabelle 4.4.: Dauer der allgemeinen Abfragen aller NoSQL-Systeme (in ms)

4.3. Gesamtauswertung

Referenzierend auf die oben angeführten Tabellen kann man sehr gut erkennen, dass die Geschwindigkeitsunterschiede zwischen relationalen Datenbanksystemen und NoSQL-Datenspeichern teils gravierend ausfallen.

Die Möglichkeiten die InfluxDB, Riak und OpenTSDB bei gleichzeitiger Skalierbarkeit und guter Performance bieten sind für derartige Zeitreihendaten wirklich geeignet. Sie können neben guter Einfügeperformance auch durch wichtige Auswertungsfunktionalitäten punkten. In Tabelle 4.4 ist im Vergleich zu Tabelle 4.2 klar erkennbar, dass hier bei den Systemen ein klarer Geschwindigkeitsvorteil in ihren Möglichkeiten vorliegt.

Cassandra zeigt im Test eine sehr lange Einfügedauer, was für den Ruf des Systems unüblich ist. Hier könnte möglicherweise eine von der Standardkonfiguration abweichende Einstellung die gewünschte Leistung ermöglichen. In Tabelle 4.4 ist allerdings eine sehr gute Abfrageperformance erkennbar.

Redis punktet durch gute Performance, allerdings ist keine wirklich passende Datenstruktur für Messdaten oä vorhanden. Auswertungen müssen daher immer in der entsprechenden Middleware durchgeführt werden. Bei den Einfügungen zeigt Hazelcast eine etwas langsamere Performance wie Redis. Vorteil hier ist allerdings, dass nach der Abfrage der Daten keine Aufbereitung der selbigen erfolgen muss. Es liegen die entsprechenden Containerobjekte vor.

Bei den dokumentbasierenden Systemen hat MongoDB klar die Nase vorn. Die Performance ist sehr gut und braucht sich auch vor Systemen die eher an die Anforderungen angepasst sind nicht verstecken. Dies sieht man vorallem in der Dauer der Einfügezeiten in Tabelle 4.3, wo in beiden Varianten MongoDB klar vorne liegt. Hier zeigt CouchDB ein annähernd konstantes Laufzeitverhalten auf hohem Niveau.

5. Fazit und Ausblick

Die Möglichkeiten um mit gesteigerten Datenmengen umzugehen sind sehr vielseitig. Hierbei kommt es immer auf den spezifischen Einsatzzweck und die an das System gestellten Anforderungen an. Eine allgemeine Empfehlung kann daher nicht pauschal getroffen werden.

Die Anwendungsformen von Big Data sind sehr vielseitig, dennoch ist derzeit das Know-How in vielen Unternehmen noch nicht ausreichend vorhanden um das volle Potenzial nutzen zu können.

Die Speicherung von Zeitreihendaten in klassischen RDBMS ist auf jeden Fall möglich, verlangt allerdings einige optimierende Schritte um die entsprechenden Datenraten zu erreichen. Bei Abfrage dieser Daten bietet dieses System allerdings einige Vorteile im Gegensatz zu den Vertretern der NoSQL-Datenbanken.

Weitere Performanceverbesserungen im Bereich der RDBMS können mit einigen optimierenden Einstellungen bei der Persistierung erreicht werden. Hierbei kommt es z.B. auf etwaige Pufferungen durch das System an. Mit

dieser können einfache Verbesserungen erreicht werden. Damit ist allerdings immer eine gewissen Unsicherheit bei der Datenintegrität verbunden, da man anfälliger für plötzliche Ausfälle des Systems wird. Diese Abwägungen müssen je nach Anwendung jeweils entsprechend getroffen werden.

Bei manche Systeme wird zur Verbesserung der Performance auf Sharding gesetzt werden können. Hierbei kann man natürlich sehr einfach wenn auch nicht ressourceneffizient eine einfache Verbesserung der Performance erreicht werden. Dieser Schritt wird allerdings nicht als erstes Mittel empfohlen, da meist zuvor einige andere weniger investitionsintensive Schritte setzen kann.

Eine Kombination mehrerer Systeme wäre hier ebenso denkbar, um gewisse Eigenschaften und Vorteile miteinander zu verbinden.

Die Tabelle 5.1 stellt im kompakten Format die Ergebnisse der Arbeit vor. Die Geschwindigkeitskriterien sind in 5 Stufen von – bis ++ gegliedert. Die Einteilung in die einzelnen Klassen erfolgte nach der Einfügezeit des 200-Sensoren-Samples. Bei der Einteilung der Abfragezeiten erfolgte eine Gesamtbetrachtung und -bewertung der Einzelergebnisse, wobei die auf allen Systemen verfügbaren Abfragen stärker gewichtet wurden. Bei der Dokumentation wurde in 3 Klassen eingeteilt. Hierbei kam es auf die frei verfügbare Dokumentation des Systems an. Abschließend wurde noch die Verfügbarkeit von kommerziellen Softwaresupport überprüft.

Name	Lizenz	Technologie	Geschw.		Skal.	Interf.	Client	Doku.	Supp.
			In	Out					
MySQL	Dual	RDBMS	~	+	✓	SQL	J/C#/Py/...	+	✓
Galera	GPL	RDBMS	-	+	✓	SQL	J/C#/Py/...	+	✓
PostgreSQL	GPL	RDBMS	~	~	✓	SQL	J/C#/Py/...	+	✓
Oracle DB	prop.	RDBMS	–	~	✓	SQL	J/C#/Py/...	~	✓
IBM DB2	prop.	RDBMS	–	-	✓	SQL	J/C#/Py/...	~	✓
Cassandra	Apache	Spalten	-	++	✓	CQL	J/C#/Py/...	+	✓
InfluxDB	MIT	Spalten/TSDB	++	++	(✓)	REST-API	✓	+	✓
CouchDB	Apache	Dokument	~	~	✓	REST-API	✓	+	✓
MongoDB	AGPL	Dokument	+	+	✓	API	J/C#/Py/...	+	✓
Hazelcast	Dual	KV-Store	+	++	✓	API	J/C#/Py/...	+	✓
Redis	BSD	KV-Store	+	++	✓	API	J/C#/Py/...	+	✓
OpenTSDB	GPL	TSDB	++	+	✓	REST-API	✓	+	✓
Riak TS	Dual	TSDB	++	++	✓	REST-API	✓	+	✓

Tabelle 5.1.: Zusammenfassende Übersichtstabelle aller Datenbanken

5. Fazit und Ausblick

Die Entwicklungen im Bereich RDBMS und Datenspeichern ist sehr starken Umbrüchen unterworfen und wird sich in Zukunft sicher nicht verlangsamen. Es entstehen einige vielversprechende Projekte, welche in ihrem Reifegrad noch nicht weit fortgeschritten sind, allerdings nichtsdestotrotz weiterer Beobachtung unterliegen sollten.

Literatur

[1] *Apache Cassandra*. URL: https://cassandra.apache.org (besucht am 01. 09. 2016) (siehe S. 43).

[2] *Apache CouchDB*. URL: http://couchdb.apache.org/ (besucht am 30. 11. 2016) (siehe S. 37).

[3] *BSON - Binary JSON*. URL: http://bsonspec.org/ (besucht am 04. 01. 2017) (siehe S. 40).

[4] *DB-Engines Ranking - die Rangliste der populärsten Datenbankmanagementsysteme*. URL: http://db-engines.com/de/ranking (besucht am 25. 11. 2016) (siehe S. 5, 21, 26, 28, 30, 39).

[5] *Edgar F. Codd - A.M. Turing Award Winner*. URL: http://amturing.acm.org/award_winners/codd_1000892.cfm (besucht am 16. 12. 2016) (siehe S. 11).

[6] W. Felter u. a. »An updated performance comparison of virtual machines and Linux containers«. In: *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. März 2015, S. 171–172. DOI: 10.1109/ISPASS.2015.7095802 (siehe S. 9).

[7] *Frequently_asked_questions - Couchdb Wiki*. URL: https://wiki.apache.org/couchdb/Frequently_asked_questions (besucht am 30. 11. 2016) (siehe S. 36, 37).

[8] H. Hu u. a. »Toward Scalable Systems for Big Data Analytics: A Technology Tutorial«. In: *IEEE Access* 2 (2014), S. 652–687. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2014.2332453 (siehe S. 2, 3).

[9] *Indexes — MongoDB Manual 3.4*. URL: https://docs.mongodb.com/manual/indexes/ (besucht am 31. 01. 2017) (siehe S. 41).

[10] *JSON and BSON | MongoDB*. URL: https://www.mongodb.com/json-and-bson (besucht am 04. 01. 2017) (siehe S. 39).

[11] Pankaj Deep Kaur und Gitanjali Sharma. »Scalable Database Management in Cloud Computing«. In: *Procedia Computer Science* 70 (2015). Proceedings of the 4th International Conference on Eco-friendly Computing and Communication Systems, S. 658–667. ISSN: 1877-0509. DOI: http://dx.doi.org/10.1016/j.procs.2015.10.102. URL: //www.sciencedirect.com/science/article/pii/S1877050915032664 (siehe S. 4).

[12] Stefanie King. *Big Data: Potential und Barrieren der Nutzung im Unternehmenskontext*. Springer, 1999. ISBN: 3-658-06585-0 (siehe S. 2).

[13] *Measuring Percona Server Docker CPU/network overhead - Percona Database Performance Blog*. URL: https://www.percona.com/blog/2016/02/05/measuring-docker-cpu-network-overhead/ (besucht am 01. 02. 2017) (siehe S. 9).

[14] *MySQL*. URL: http://www.mysql.com/ (besucht am 25. 11. 2016) (siehe S. 21).

[15] *MySQL :: Legal Policies*. URL: http://www.mysql.com/about/legal/ (besucht am 30. 11. 2016) (siehe S. 21).

[16] *Oracle Completes Acquisition of Sun*. URL: http://www.oracle.com/us/corporate/press/044428 (besucht am 30. 11. 2016) (siehe S. 21).

[17] L. Ou u. a. »Chapter 12 - Security and Privacy in Big Data«. In: *Big Data*. Hrsg. von Rajkumar Buyya, Rodrigo N. Calheiros und Amir Vahid Dastjerdi. Morgan Kaufmann, 2016, S. 285–308. ISBN: 978-0-12-805394-2. DOI: http://dx.doi.org/10.1016/B978-0-12-805394-2.00012-X. URL: http://www.sciencedirect.com/science/article/pii/B978012805394200012X (siehe S. 3).

[18] *Partitioning: how to split data among multiple Redis instances. – Redis*. URL: https://redis.io/topics/partitioning (besucht am 25. 01. 2017) (siehe S. 49).

[19] *PostgreSQL: History*. URL: https://www.postgresql.org/about/history/ (besucht am 30. 11. 2016) (siehe S. 26).

[20] *PostgreSQL: License*. URL: https://www.postgresql.org/about/licence/ (besucht am 30. 11. 2016) (siehe S. 26).

[21] Tilmann Rabl, Michael Frank und Manuel Danisch. *NoSQL Performance Test - In-Memory Performance Comparison of SequoiaDB, Cassandra, and MongoDB*. 2014 (siehe S. 45).

[22] *RIAK TS Technical Overview - Whitepaper*. URL: http://info.basho.com/rs/721-DGT-611/images/RiakTS-Enterprise-Technical-Overview.PDF (besucht am 25. 01. 2017) (siehe S. 57).

[23] *State of MongoDB March, 2010 | MongoDB*. URL: https://www.mongodb.com/blog/post/state-of-mongodb-march-2010 (besucht am 01. 12. 2016) (siehe S. 39).

[24] Jonathan Stuart Ward und Adam Barker. »Undefined By Data: A Survey of Big Data Definitions«. In: *CoRR* abs/1309.5821 (2013). URL: http://arxiv.org/abs/1309.5821 (siehe S. 2).

[25] *White paper: Minimizing downtime and maximizing elasticity with Galera Cluster for MySQL*. Techn. Ber. codership.com, 2013. URL: http://galeracluster.com/wp-content/uploads/2013/10/Minimizing-downtime-and-maximizing-elasticity-with-Galera-Cluster-for-MySQL.pdf (siehe S. 24).

This document is set in Palatino, compiled with pdfLaTeX2e and Biber.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Appendix

Anhang A.

SQL

Diese Klasse hat die Aufgabe die Tests für das Datenmodell Variante 2 für SQL auszuführen. Dabei werden die datenbankspezifischen Werte über eine Konfiguration eingespielt, welche sich in Datentypdefinition und Treiber unterscheidet.

A.1. MultiInsertionThread

```
1 package at.tugraz.db.client;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
```

```
6  import java.sql.Statement;
7  import java.util.Locale;
8
9  public class MultiInsertionThread extends DbThread {
10
11     protected final String dataTypeKey;
12     protected final String dataTypeData;
13     protected final String connectionUrl;
14
15     protected final int times;
16     protected final boolean autoCommit;
17     protected final int interval;
18     protected String dataTypeTime;
19     protected boolean create = true;
20     protected boolean cleanup = true;
21     protected String username;
22     protected String password;
23     protected String facility;
24     protected int offset;
25
26     public MultiInsertionThread(DatabaseConfig dbConfig) {
27         super(dbConfig.getInsertionSamples(),
   ↪ dbConfig.getMaxSensors(), dbConfig.isAutoCommit(),
   ↪ dbConfig.getTable());
```

```
28          this.dataTypeKey = dbConfig.getDataTypeKey();
29          this.dataTypeData = dbConfig.getDataTypeData();
30          this.connectionUrl = dbConfig.getConnectionUrl();
31          this.autoCommit = dbConfig.isAutoCommit();
32          this.times = dbConfig.getTimes();
33          this.interval = dbConfig.getInterval();
34          this.username = dbConfig.getUsername();
35          this.password = dbConfig.getPassword();
36          this.cleanup = dbConfig.isCleanup();
37          this.create = dbConfig.isCreate();
38          this.dataTypeTime = dbConfig.getDataTypeTime();
39          this.offset = dbConfig.getOffset();
40          this.facility = dbConfig.getFacility();
41      }
42
43      public boolean isCleanup() {
44          return cleanup;
45      }
46
47      public void setCleanup(boolean cleanup) {
48          this.cleanup = cleanup;
49      }
50
51      @Override
```

```
52      public void run() {
53          try (Connection conn =
    ↪ DriverManager.getConnection(connectionUrl, username,
    ↪ password)) {
54                  {
55                      conn.setAutoCommit(autoCommit);
56                      Statement stmt = null;
57                      String sql;
58
59                      long start;
60                      for (long k = interval; k <= maxSensors; k = k +
    ↪ interval) {
61
62                          String vals = "";
63                          for (long i = 0; i < k; i++) {
64                              vals = vals + ", value" + i + " " +
    ↪ dataTypeData;
65                          }
66                          if (create) {
67                              stmt = conn.createStatement();
68                              stmt.execute("DROP TABLE IF EXISTS "+
    ↪ table);
69                              stmt.execute("CREATE TABLE  " + table +
    ↪ " (aid "+dataTypeKey+", ti " + dataTypeTime + vals + ")");
```

```
70                          stmt.execute("ALTER TABLE " + table + "
    ↪ add primary key (ti, aid)");
71                          stmt.close();
72                          if(!autoCommit)
73                              conn.commit();
74                      }
75                      System.out.print(k);
76                      for (int m = 0; m < times; m++) {
77
78                          start = System.currentTimeMillis();
79                          stmt = conn.createStatement();
80
81                          for (long i = 1+offset; i <=
    ↪ insertionSamples; i++) {
82                              vals = "";
83                              for (long j = 0; j < k; j++) {
84
85                                  vals += ", " +
    ↪ String.format(Locale.ENGLISH, "%.4f",
86                                          (Math.random() * 100 +
    ↪ 1));
87                              }
88                              sql = "INSERT INTO " + table + "
    ↪ VALUES ("+facility+"," + System.nanoTime() + vals + ")";
```

```
89                              stmt.executeUpdate(sql);
90
91                              if ((i % 10000) == 0) {
92                                  if(!autoCommit)
93                                      conn.commit();
94                              }
95                          }
96                          stmt.close();
97                          if (!autoCommit)
98                              conn.commit();
99
100                         System.out.print(";" +
    ↪ (System.currentTimeMillis() - start));
101                         if (cleanup) {
102
103                             sql = "DELETE FROM " + table;
104                             stmt = conn.createStatement();
105                             stmt.executeUpdate(sql);
106                             stmt.close();
107                             if (!autoCommit)
108                                 conn.commit();
109                         }
110                     }
111                     System.out.println("");
```

```
112                     if (cleanup) {
113                         stmt = conn.createStatement();
114                         stmt.execute("DROP TABLE " + table);
115 
116                         stmt.close();
117                         if (!autoCommit)
118                             conn.commit();
119                     }
120                 }
121                 stmt.close();
122 
123             }
124         } catch (SQLException e) {
125             e.printStackTrace();
126         }
127     }
128 }
```

A.2. SingleInsertionThread

```
1 package at.tugraz.db.client;
2 
3 import java.sql.Connection;
```

```
4  import java.sql.DriverManager;
5  import java.sql.SQLException;
6  import java.sql.Statement;
7  import java.util.Locale;
8
9
10 public class SingleInsertionThread extends DbThread {
11
12     protected final int interval;
13
14
15     protected final int times;
16     protected final String connectionString;
17     protected final String dataTypeData, dataTypeKey,
   ↪  dataTypeTime, username, password;
18     protected boolean create = true;
19     protected boolean cleanup = true;
20     protected int offset;
21     protected String facility;
22
23     public SingleInsertionThread(DatabaseConfig dbConfig) {
24         super(dbConfig.getInsertionSamples(),
   ↪  dbConfig.getMaxSensors(), dbConfig.isAutoCommit(),
   ↪  dbConfig.getTable());
```

```
25          this.dataTypeKey = dbConfig.getDataTypeKey();
26          this.dataTypeData = dbConfig.getDataTypeData();
27          this.connectionString = dbConfig.getConnectionUrl();
28          this.times = dbConfig.getTimes();
29          this.interval = dbConfig.getInterval();
30          this.username = dbConfig.getUsername();
31          this.password = dbConfig.getPassword();
32          this.cleanup = dbConfig.isCleanup();
33          this.create = dbConfig.isCreate();
34          this.dataTypeTime = dbConfig.getDataTypeTime();
35          this.offset = dbConfig.getOffset();
36          this.facility = dbConfig.getFacility();
37      }
38
39      @Override
40      public void run() {
41          Statement statement = null;
42
43
44          try (Connection connection =
    ↪ DriverManager.getConnection(connectionString, username,
    ↪ password)) {
45              connection.setAutoCommit(autoCommit);
46              if (create) {
```

```
47                  statement = connection.createStatement();
48                  statement.execute("DROP TABLE IF EXISTS " +
    ↪ table+";");
49                  statement.execute("CREATE TABLE " + table + "
    ↪ (id " + dataTypeKey + ", aid " + dataTypeKey + ",  ti " +
    ↪ dataTypeTime + ", value " + dataTypeData + ")");
50                  statement.execute("ALTER TABLE " + table + " add
    ↪ primary key (id,ti, aid)");
51                  statement.close();
52                  if (!autoCommit)
53                      connection.commit();
54              }
55              String sql;
56
57              long startcd;
58              for (long k = interval; k <= maxSensors; k = k +
    ↪ interval) {
59                  System.out.print(k);
60                  for (int m = 0; m < times; m++) {
61                      start = System.currentTimeMillis();
62                      statement = connection.createStatement();
63
64                      for (long i = 1+offset; i <=
    ↪ insertionSamples; i++) {
```

```
65                          for (long j = 0; j < k; j++) {
66                              ;
67                              sql = "INSERT INTO " + table + "
    ↪ VALUES (" + j + ", "+facility+", " + System.nanoTime() +
68                                      ", " +
    ↪ String.format(Locale.ENGLISH, "%.4f", Math.random() * 100 +
    ↪ 1) + ")";
69
70                              statement.executeUpdate(sql);
71                          }
72                          if (i % 10000 == 0)
73                              if (!autoCommit)
74                                  connection.commit();
75                      }
76
77
78                      statement.close();
79                      if (!autoCommit)
80                          connection.commit();
81                      System.out.print(";" +
    ↪ (System.currentTimeMillis() - start));
82                      if (cleanup) {
83                          cleanTable(connection);
84                      }
```

```
85                  }
86                  System.out.println("");
87              }
88              if (cleanup) {
89                  dropTable(connection);
90              }
91
92
93          } catch (SQLException e) {
94              e.printStackTrace();
95          }
96
97          System.out.println(System.currentTimeMillis() - start);
98      }
99
100
101 }
```

Anhang B.

Dokumentorientierte Datenbanken

B.1. CouchDB

B.1.1. CouchDBClientThread

```
1 package at.tugraz;
2
3 import at.tugraz.table.MeasData;
4 import at.tugraz.table.MeasData50;
5 import at.tugraz.table.MeasDataAll;
6 import org.ektorp.CouchDbConnector;
```

```
7  import org.ektorp.CouchDbInstance;
8  import org.ektorp.ViewQuery;
9  import org.ektorp.http.HttpClient;
10 import org.ektorp.http.StdHttpClient;
11 import org.ektorp.impl.StdCouchDbConnector;
12 import org.ektorp.impl.StdCouchDbInstance;
13 import org.ektorp.support.View;
14
15
16 import java.net.MalformedURLException;
17 import java.util.List;
18 import java.util.UUID;
19
20 public class CouchDBClientThread extends Thread {
21     @Override
22     public void run() {
23
24         HttpClient httpClient;
25         try {
26             httpClient = new
   ↪ StdHttpClient.Builder().url("http://localhost:5984")
27                     .build();
28         } catch (MalformedURLException e) {
29             e.printStackTrace();
```

```
30              return;
31          }
32
33          CouchDbInstance dbInstance = new
    ↪ StdCouchDbInstance(httpClient);
34          CouchDbConnector db = new StdCouchDbConnector("testdb",
    ↪ dbInstance);
35
36          db.createDatabaseIfNotExists();
37          for (int times = 0; times < 10; times++) {
38              long start = System.currentTimeMillis();
39              for (int i = 0; i < 1_000; i++) {
40                  db.create(UUID.randomUUID().toString(), new
    ↪ MeasDataAll(System.currentTimeMillis(), "gen1"));
41              }
42              System.out.println(System.currentTimeMillis() -
    ↪ start);
43          }
44
45          long start = System.currentTimeMillis();
46          ViewQuery v = new ViewQuery().allDocs();
47          List<MeasDataAll> list = db.queryView(v,
    ↪ MeasDataAll.class);
48          System.out.println(System.currentTimeMillis() - start);
```

```
49          System.out.println(list.size());
50
51      }
52  }
```

B.1.2. CouchDBQueryThread

```
1  package at.tugraz;
2
3
4  public class CouchDBQueryThread {
5      public static void main(String[] args) {
6
7      }
8  }
```

B.2. MongoDB

B.2.1. MongoDBSingleInsertThread

```
1  package at.tugraz.db.client;
2
3  import com.mongodb.MongoClient;
```

```
4  import com.mongodb.client.MongoCollection;
5  import com.mongodb.client.MongoDatabase;
6  import org.bson.Document;
7
8  import java.util.Locale;
9
10
11 public class MongoDbSingleInsertThread extends DbThread {
12
13     int interval = 10;
14     int facility = 1;
15
16     String facilityName = "";
17     private int iterations;
18     private boolean cleanup;
19
20     public MongoDbSingleInsertThread(int insertion_samples, int
   ↪  maxSensors){
21         super(insertion_samples, maxSensors,true, "dataDB1");
22         this.facilityName = "test1";
23         this.iterations = 10;
24         this.cleanup = true;
25     }
26
```

```
27      public MongoDbSingleInsertThread(int insertion_samples, int
    ↪ maxSensors, int facility, int iterations, boolean cleanup) {
28          super(insertion_samples, maxSensors, true, "dataDB1");
29          this.facility = facility;
30          this.iterations = iterations;
31          this.cleanup = cleanup;
32          facilityName = "test" + facility;
33      }
34
35      @Override
36      public void run() {
37          MongoClient mongoClient = new MongoClient();
38          MongoDatabase db = mongoClient.getDatabase("test");
39          MongoCollection<Document> coll =
    ↪ db.getCollection("dataDB1");
40          long start;
41          for (int m = interval; m <= maxSensors; m += interval) {
42              System.out.print(m);
43              for (int k = 0; k < iterations; k++) {
44                  start = System.currentTimeMillis();
45                  for (long i = 1; i <= insertionSamples; i++) {
46                      for (long j = 0; j < m; j++) {
47                          coll.insertOne(
```

```
48                                  new Document("Facility",
    ↪ facilityName)
49                                      .append("Sensor" + j,
    ↪ String.format(Locale.ENGLISH, "%.4f",
50                                              Math.random() *
    ↪ 100 + 1))
51                                      .append("Time", (new
    ↪ Long(i)).toString()));
52
53                      }
54                  }
55                  System.out.print(";" +
    ↪ (System.currentTimeMillis() - start));
56
57                  if(cleanup)
58                      coll.deleteMany(new Document());
59              }
60              System.out.println("");
61          }
62
63          mongoClient.close();
64      }
65  }
```

B.2.2. MongoDBMultiInsertThread

```
1  package at.tugraz.db.client;
2
3  import com.mongodb.MongoClient;
4  import com.mongodb.client.MongoCollection;
5  import com.mongodb.client.MongoDatabase;
6  import org.bson.Document;
7
8  import java.util.Locale;
9
10 public class MongoDbMultiInsertThread extends DbThread {
11
12     int interval = 10;
13
14     public MongoDbMultiInsertThread(int insertion_samples, long
   ↪  maxSensors) {
15         super(insertion_samples, maxSensors, true, "dataDB2");
16     }
17
18     @Override
19     public void run() {
20         MongoClient mongoClient = new MongoClient();
21         MongoDatabase db = mongoClient.getDatabase("test");
```

```
22          MongoCollection<Document> coll =
    ↪ db.getCollection("dataDB2");
23
24          long start;
25          Document doc;
26          for (int m = interval; m <= maxSensors; m += interval) {
27              System.out.print(m);
28              for (int k = 0; k < 10; k++) {
29                  start = System.currentTimeMillis();
30                  for (long i = 1; i <= 1000; i++) {
31                      doc = new Document("Facility", "test1")
32                              .append("Time", (new
    ↪ Long(i)).toString());
33                      for (long j = 0; j < m; j++) {
34                          doc.append("Sensor" + j,
    ↪ String.format(Locale.ENGLISH, "%.4f",
35                                  Math.random() * 100 + 1));
36                      }
37                      coll.insertOne(doc);
38                  }
39                  System.out.print(";" +
    ↪ (System.currentTimeMillis() - start));
40                  coll.deleteMany(new Document());
41              }
```

```
42                 System.out.println("");
43             }
44             mongoClient.close();
45         }
46     }
```

B.2.3. MongoDBQueryThread

```
1  package at.tugraz.db.client;
2
3
4  import com.mongodb.Block;
5  import com.mongodb.MongoClient;
6  import com.mongodb.client.FindIterable;
7  import com.mongodb.client.MongoCollection;
8  import com.mongodb.client.MongoDatabase;
9  import org.bson.Document;
10
11 import java.util.concurrent.atomic.AtomicLong;
12
13 import static com.mongodb.client.model.Filters.*;
14
15 public class MongoDbQueryThread extends DbQueryThread {
16
```

```
17      @Override
18      public void run() {
19          MongoClient mongoClient = new MongoClient();
20          MongoDatabase db = mongoClient.getDatabase("test");
21          MongoCollection<Document> coll1 =
    ↪ db.getCollection("dataDB2");
22
23          final AtomicLong counter = new AtomicLong();
24          long start;
25
26          start = System.currentTimeMillis();
27          long elemCount = coll1.count();
28          System.out.println("query 1: " + elemCount + ";" +
    ↪ (System.currentTimeMillis()-start));
29
30
31          start = System.currentTimeMillis();
32          FindIterable<Document> iterable = coll1.find(
33                  and(eq("Facility", "anlage35"), eq("Sensor",
    ↪ "sensor76")));
34          elemCount = System.currentTimeMillis() - start;
35          counter.set(0);
36          iterable.forEach((Block<Document>) document ->
    ↪ counter.incrementAndGet());
```

```
37          System.out.println("query 2: " + counter.get() + ";" +
38                  (System.currentTimeMillis()-start) + "(" +
    ↪ elemCount +")");
39
40
41          start = System.currentTimeMillis();
42          iterable = coll1.find(
43                  and(eq("Facility", "anlage77"),and(gt("Time",
44                          9500),lt("Time",9520))));
45          elemCount = System.currentTimeMillis() - start;
46          counter.set(0);
47          iterable.forEach((Block<Document>) document ->
    ↪ counter.incrementAndGet());
48          System.out.println("query 3: " +
49                  counter.get() + ";" +
    ↪ (System.currentTimeMillis()-start) + "(" + elemCount +")");
50      }
51
52
53  }
```

Anhang C.

Spaltenorientierte Datenbanken

C.1. Cassandra

```
1  package at.tugraz;
2
3
4  import com.datastax.driver.core.*;
5
6  import java.math.BigInteger;
7  import java.text.SimpleDateFormat;
8  import java.util.Date;
9  import java.util.concurrent.BrokenBarrierException;
10 import java.util.concurrent.CyclicBarrier;
```

```
11
12 public class CassandraClientThread extends Thread {
13
14     private Cluster cluster;
15     private Session session;
16     private String sensorId;
17     private PreparedStatement query;
18
19     private boolean doLoad = true;
20     private boolean doQuery = true;
21
22
23     public CassandraClientThread(String sensorId) {
24         this.sensorId = sensorId;
25     }
26
27
28     @Override
29     public void run() {
30         this.connect("localhost");
31         if (doLoad) {
32             this.createSchema();
33             this.loadData();
34         }
```

```
35          if (doQuery)
36              this.queryData();
37          this.cleanup();
38          cluster.close();
39      }
40
41
42      private void connect(String node) {
43          this.cluster =
    ↪ Cluster.builder().addContactPoint(node).build();
44          Metadata metadata = cluster.getMetadata();
45          System.out.printf("Cluster: %s\n",
    ↪ metadata.getClusterName());
46
47          for (Host h : metadata.getAllHosts()) {
48              System.out.printf("Datacenter: %s, Host: %s, Rack:
    ↪ %s\n", h.getDatacenter(), h.getAddress(), h.getRack());
49          }
50
51          session = cluster.connect();
52      }
53
54      private void createSchema() {
```

```
55          session.execute("CREATE KEYSPACE IF NOT EXISTS
    ↪ testschema WITH replication={'class':'SimpleStrategy',
    ↪ 'replication_factor': 1};");
56          session.execute("CREATE TABLE IF NOT EXISTS
    ↪ testschema.databyday (sensor_id text, date text, event_time
    ↪ timestamp, measvalue double, PRIMARY KEY((sensor_id, date),
    ↪ event_time, sensor_id));");
57      }
58
59      private void loadData() {
60
61          SimpleDateFormat dateFormatter = new
    ↪ SimpleDateFormat("yyyy-MM-dd");
62          SimpleDateFormat timeFormatter = new
    ↪ SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");
63
64          for (int ges = 1; ges <= 1; ges += 10) {
65              System.out.print(ges);
66              for (int it = 0; it < 10; it++) {
67                  long start = System.currentTimeMillis();
68                  for (int i = 0; i < 20000; i++) {
69                      Date dNow = new Date();
70                      for (int sen = 0; sen < ges; sen++)
```

```
71                          this.loadDataCQL(sensorId + sen,
    ↪ dateFormatter.format(dNow), timeFormatter.format(dNow),
    ↪ "2.23333");
72                  }
73                  System.out.print(";" +
    ↪ (System.currentTimeMillis() - start));
74              }
75              //session.execute("TRUNCATE testschema.databyday");
76              System.out.println("");
77          }
78      }
79
80      private void loadDataCQL(String sensorId, String date,
    ↪ String event_time, String value) {
81          StringBuilder sb = new StringBuilder();
82          sb.append("INSERT INTO testschema.databyday(sensor_id,
    ↪ date, event_time, measvalue) VALUES ('");
83          sb.append(sensorId).append("','").append(date)
84          .append("','").append(event_time).append("',");
85          sb.append(value).append(");");
86
87          session.execute(sb.toString());
88      }
89
```

```
90      private void cleanup() {
91          session.execute("DROP TABLE testschema.databyday");
92      }
93
94      private void queryData() {
95          long start = System.currentTimeMillis();
96          ResultSet result = session.execute("SELECT Count(*) FROM
    ↪ testschema.databyday where sensor_id = 'sensor1' and date =
    ↪ '2017-0'");
97          System.out.println("query1: " +
    ↪ (System.currentTimeMillis() - start));
98
99          start = System.currentTimeMillis();
100         result = session.execute("SELECT * FROM
    ↪ testschema.databyday");
101         System.out.println("query2: " +
    ↪ (System.currentTimeMillis() - start));
102
103
104         start = System.currentTimeMillis();
105         result = session.execute("SELECT * FROM
    ↪ testschema.databyday WHERE event_time < 1485769416100 AND
    ↪ event_time > 1485769382196 ALLOW FILTERING");
```

```
106         System.out.println("query3: " +
    ↪ (System.currentTimeMillis() - start));
107     }
108 }
```

C.2. InfluxDB

C.2.1. InfluxInsertionThread

```
1  package at.tugraz;
2
3  import at.tugraz.db.client.DatabaseConfig;
4  import org.influxdb.InfluxDB;
5  import org.influxdb.InfluxDBFactory;
6  import org.influxdb.dto.BatchPoints;
7  import org.influxdb.dto.Point;
8
9  import java.util.concurrent.TimeUnit;
10
11 public class InfluxInsertionThread2 extends Thread {
12
13     protected String dbName;
14     private boolean cleanup;
```

```
15      private String connectionUrl, username, password;
16      private DatabaseConfig config;
17
18      public InfluxInsertionThread2(DatabaseConfig config) {
19          this.cleanup = config.isCleanup();
20          this.dbName = config.getTable();
21          this.connectionUrl = config.getConnectionUrl();
22          this.username = config.getUsername();
23          this.password = config.getPassword();
24          this.config = config;
25      }
26
27      @Override
28      public void run() {
29          InfluxDB db =
    ↪ InfluxDBFactory.connect(config.getConnectionUrl(),
    ↪ config.getUsername(), config.getPassword());
30          db.createDatabase(config.getDatabase());
31
32          BatchPoints bp =
    ↪ BatchPoints.database(config.getDatabase()).retentionPolicy("autogen").consist
33
```

```
34          for (int totalSensors = config.getInterval();
    ↪ totalSensors <= config.getMaxSensors(); totalSensors +=
    ↪ config.getInterval()) {
35              System.out.print(totalSensors);
36
37              for (int it = 0; it < config.getTimes(); it++) {
38                  long start = System.currentTimeMillis();
39                  for (int i = 0; i <
    ↪ config.getInsertionSamples(); i++) {
40                      long time = System.nanoTime();
41                      for (int sen = 1; sen <= totalSensors;
    ↪ sen++)
42
    ↪ bp.point(Point.measurement("anlage1").addField("sensor" +
    ↪ sen, Math.random() * 100).time(time,
    ↪ TimeUnit.NANOSECONDS).build());
43
44                      if (i != 0 && ((i * totalSensors) % 10000)
    ↪ == 0) {
45                          db.write(bp);
46                          bp.getPoints().clear();
47                      }
48                  }
49
```

```
50                  db.write(bp);
51
52
53                  System.out.print(';');
54                  System.out.print(System.currentTimeMillis() -
    ↪ start);
55              }
56              System.out.println("");
57              //QueryResult query1 = db.query(new Query("SELECT *
    ↪ from anlage1", dbName));
58
59              if (cleanup) {
60                  db.deleteDatabase(dbName);
61                  try {
62                      Thread.sleep(2000);
63                  } catch (InterruptedException e) {
64                      e.printStackTrace();
65                  }
66                  db.createDatabase(dbName);
67                  try {
68                      Thread.sleep(2000);
69                  } catch (InterruptedException e) {
70                      e.printStackTrace();
71                  }
```

```
72                 }
73                 //System.out.println(query1);
74             }
75
76             if (cleanup) {
77                 db.deleteDatabase(dbName);
78                 System.out.println("deleted");
79             }
80         }
81 }
```

C.2.2. InfluxQueryThread

```
1  package at.tugraz;
2
3  import at.tugraz.db.client.DatabaseConfig;
4  import org.influxdb.InfluxDB;
5  import org.influxdb.InfluxDBFactory;
6  import org.influxdb.dto.Query;
7  import org.influxdb.dto.QueryResult;
8
9  public class InfluxQueryThread extends Thread {
10
11     DatabaseConfig conf;
```

```
12
13      public InfluxQueryThread(DatabaseConfig dbconfig) {
14          conf = dbconfig;
15      }
16
17      @Override
18      public void run() {
19          String q1 = "Select count(sensor1) FROM anlage1";
20          String q2 = "Select * from anlage1";
21          String q3 = "Select * from anlage1 where time >
    ↪ 1212121";
22
23          InfluxDB db =
    ↪ InfluxDBFactory.connect(conf.getConnectionUrl(),
    ↪ conf.getUsername(), conf.getPassword());
24
25          String dbName = "testdb";
26
27          long start1 = System.currentTimeMillis();
28          QueryResult query = db.query(new Query(q1, dbName));
29          System.out.println(query.getResults());
30          long dur1 = System.currentTimeMillis() - start1;
31
32          long start2 = System.currentTimeMillis();
```

```
33            query = db.query(new Query(q2, dbName));
34            //System.out.println(query);
35            long dur2 = System.currentTimeMillis() - start2;
36
37            long start3 = System.currentTimeMillis();
38            query = db.query(new Query(q3, dbName));
39            //System.out.println(query);
40            long dur3 = System.currentTimeMillis() - start3;
41
42            System.out.println(dur1 + " " + dur2+ " " + dur3);
43
44        }
45    }
```

Anhang D.

Key-Value-Stores

D.1. Redis

```
1 import com.lambdaworks.redis.api.sync.RedisKeyCommands;
2 import com.lambdaworks.redis.api.sync.RedisSortedSetCommands;
3
4 import java.util.Locale;
5 import java.util.concurrent.BrokenBarrierException;
6 import java.util.concurrent.CyclicBarrier;
7
8 public class RedisSingleTestThread extends Thread {
```

```
9       RedisKeyCommands<String, String>
    ↪   clean;RedisSortedSetCommands<String, String>
    ↪   cmd;CyclicBarrier cyclicBarrier;
10      String measId;
11
12      public RedisSingleTestThread(String objectId, String
    ↪   sensorId, CyclicBarrier cyclicBarrier) {
13          this.measId = objectId + "_" + sensorId;
14          this.cyclicBarrier = cyclicBarrier;
15      }
16
17      @Override
18      public void run() {
19          try {
20              System.out.println("start thread");
21              cyclicBarrier.await();
22              StringBuilder vals;
23              double random = 0;
24              long lStartTime = 0, lEndTime = 0;
25              for (int ins = 1; ins <= 1; ins += 10) {
26                  System.out.print(ins);
27                  for (int run = 0; run < 1; run++) {
28                      lStartTime = System.currentTimeMillis();
29                      for (int i = 0; i < 1000; i++) {
```

```
30                      for (int v = 0; v < ins; v++) {
31                          random = Math.random() * 100 + 1;
32                          cmd.zadd(measId,
    ↪ System.currentTimeMillis(), String.format(Locale.ENGLISH,
    ↪ "%.4f", random));
33                      }
34                  }
35                  lEndTime = System.currentTimeMillis();
36                  System.out.print(";" + (lEndTime -
    ↪ lStartTime));
37                  for (int v = 0; v < ins; v++)
38                      clean.del(measId + v);
39              }
40              System.out.println("");
41          }
42          cyclicBarrier.await();
43      } catch (InterruptedException e) {
44          e.printStackTrace();
45      } catch (BrokenBarrierException e) {
46          e.printStackTrace();
47      }
48
49
50  }
```

```
51 }
```

D.2. Hazelcast

```
1  package at.tugraz;
2
3
4  import com.hazelcast.core.MultiMap;
5
6  import java.util.Collection;
7  import java.util.concurrent.BrokenBarrierException;
8  import java.util.concurrent.CyclicBarrier;
9
10 public class HazelcastMultimapClientThread extends
   ↪ HazelcastClientThread {
11
12     public HazelcastMultimapClientThread(String name, int
   ↪ iterations) {
13         super(name, iterations);
14     }
15
16     @Override
17     public void run() {
```

```
18
19          for(int sensors = 50; sensors <= 200; sensors+=50 ) {
20              System.out.print(sensors);
21              for (int times = 0; times < 10; times++) {
22                  long insert_start = System.currentTimeMillis();
23
24              for (int i = 0; i < sensors; i++) {
25                  MultiMap<String, DataTuple> bla =
    ↪ instance.getMultiMap("sensor" + i);
26
27
28                  for (int j = 0; j < iterations; j++) {
29                      bla.put("data170110", new
    ↪ DataTuple(System.currentTimeMillis(), Math.random() * 100));
30                  }
31              }
32
33              System.out.print(";" + (System.currentTimeMillis() -
    ↪ insert_start));
34              }
35              System.out.println("");
36          }
37          MultiMap<String, DataTuple> bla =
    ↪ instance.getMultiMap("sensor25");
```

```
38
39          long start = System.currentTimeMillis();
40          Collection<DataTuple> data = bla.get("data170110");
41          System.out.println(data.size());
42          System.out.println(System.currentTimeMillis() - start);
43          instance.shutdown();
44      }
45
46
47  }
```

Anhang E.

Zeitreihendatenbanken

E.1. OpenTSDB

E.1.1. OTSDBInsertionThread

```
1 package at.tugraz;
2
3
4 import com.google.common.collect.Lists;
5 import com.google.common.net.HttpHeaders;
6 import com.google.gson.Gson;
7 import org.apache.http.Header;
8 import org.apache.http.HttpResponse;
```

```
9  import org.apache.http.client.ClientProtocolException;
10 import org.apache.http.client.HttpClient;
11 import org.apache.http.client.methods.HttpPost;
12 import org.apache.http.entity.ContentType;
13 import org.apache.http.entity.StringEntity;
14 import org.apache.http.impl.client.DefaultHttpClient;
15 import org.apache.http.impl.client.HttpClientBuilder;
16 import org.apache.http.impl.client.HttpClients;
17 import org.apache.http.message.BasicHeader;
18
19 import java.io.BufferedReader;
20 import java.io.IOException;
21 import java.io.InputStreamReader;
22 import java.io.UnsupportedEncodingException;
23 import java.text.SimpleDateFormat;
24 import java.util.ArrayList;
25 import java.util.Arrays;
26 import java.util.Date;
27 import java.util.List;
28 import java.util.concurrent.BrokenBarrierException;
29 import java.util.concurrent.CyclicBarrier;
30 import java.util.stream.Collectors;
31
32 public class OTSDBInsertionThread extends Thread {
```

```
33      private int messages;
34
35
36      public OTSDBInsertionThread(int messages){
37          this.messages = messages;
38      }
39
40
41      @Override
42      public void run() {
43          try {
44              Gson gson = new Gson();
45              Header header = new
    ↪ BasicHeader(HttpHeaders.CONTENT_TYPE, "application/json");
46              List<Header> headers = Lists.newArrayList(header);
47              HttpClient client =
    ↪ HttpClients.custom().setDefaultHeaders(headers).build();
48              HttpPost post = new
    ↪ HttpPost("http://localhost:4242/api/put");
49              List<OTSDBValueContainer> f = new
    ↪ ArrayList<>(messages);
50              long start = System.currentTimeMillis();
51              for (int i = 0; i < messages; i++) {
52                  long timestamp = System.currentTimeMillis();
```

```
53                  float b =  (float) Math.random() * 100 + 1;
54                  OTSDBValueContainer a = new
    ↪ OTSDBValueContainer("sensor1", timestamp, b);
55                  f.add(a);
56                  if(i != 0 && i % 50 == 0){
57                      String result = "[" + f.stream().map(x ->
    ↪ gson.toJson(x)).collect(Collectors.joining(",")) + "]";
58                      post.setEntity(new StringEntity(result,
    ↪ ContentType.APPLICATION_JSON));
59                      client.execute(post);
60
61                      f.clear();
62                  }
63              }
64              if(!f.isEmpty()) {
65                  String result = "[" + f.stream().map(x ->
    ↪ gson.toJson(x)).collect(Collectors.joining(",")) + "]";
66                  post.setEntity(new StringEntity(result,
    ↪ ContentType.APPLICATION_JSON));
67                  HttpResponse response = client.execute(post);
68
69              }
70
71              System.out.print(";");
```

```
72              System.out.print(System.currentTimeMillis() -
    ↪  start);
73          } catch (ClientProtocolException e) {
74              e.printStackTrace();
75          } catch (IOException e) {
76              e.printStackTrace();
77          }
78      }
79  }
```

E.1.2. OTSDBQueryThread

```
1  package at.tugraz;
2
3
4  import com.google.gson.Gson;
5  import org.apache.commons.io.IOUtils;
6  import org.apache.http.HttpRequest;
7  import org.apache.http.HttpResponse;
8  import org.apache.http.client.HttpClient;
9  import org.apache.http.client.methods.HttpGet;
10 import org.apache.http.impl.client.HttpClients;
11
12 import java.io.FileWriter;
```

```
13 import java.io.IOException;
14 import java.io.StringWriter;
15 import java.util.ArrayList;
16 import java.util.List;
17
18 public class OTSDBQueryThread extends Thread {
19
20     @Override
21     public void run() {
22
23         List<HttpGet> getList = new ArrayList<>(4);
24         getList.add(new
    ↪ HttpGet("http://localhost:4242/api/query?start=2016/12/02-1
    ↪ 0:39:46&end=2017/01/30-11:05:26&m=sum:sensor1"));
25         getList.add(new
    ↪ HttpGet("http://localhost:4242/api/query?start=2016/12/02-1
    ↪ 0:39:46&end=2017/01/30-11:05:26&m=count:sensor1"));
26         getList.add(new
    ↪ HttpGet("http://localhost:4242/api/query?start=2016/12/02-1
    ↪ 0:39:46&end=2017/01/30-11:05:26&m=min:sensor1"));
27         getList.add(new
    ↪ HttpGet("http://localhost:4242/api/query?start=2016/12/02-1
    ↪ 0:39:46&end=2017/01/30-11:05:26&m=avg:sensor1"));
28
```

```
29          int i = 1;
30
31          for (HttpGet get : getList) {
32              try {
33                  HttpClient client = HttpClients.createDefault();
34                  long start = System.currentTimeMillis();
35                  HttpResponse execute = client.execute(get);
36                  System.out.println(execute.getStatusLine());
37                  StringWriter sw = new StringWriter();
38                  IOUtils.copy(execute.getEntity().getContent(),
    ↪ sw);
39                  String result = sw.toString();
40
41                  System.out.println("done" + (i++) + ": " +
    ↪ (System.currentTimeMillis() - start));
42
43                  FileWriter fw = new
    ↪ FileWriter("/home/oliver/otsdb_res_"+i+".txt");
44                  fw.write(result);
45                  fw.close();
46
47                  Thread.sleep(1000);
48              } catch (IOException e) {
49                  e.printStackTrace();
```

```
50                } catch (InterruptedException e) {
51                    e.printStackTrace();
52                }
53            }
54
55        }
56    }
```

E.2. Riak TS

E.2.1. RiakTsClientThread

```
1  package at.tugraz;
2
3  import com.basho.riak.client.api.RiakClient;
4  import com.basho.riak.client.api.commands.timeseries.Delete;
5  import com.basho.riak.client.api.commands.timeseries.Query;
6  import com.basho.riak.client.api.commands.timeseries.Store;
7  import com.basho.riak.client.core.RiakFuture;
8  import com.basho.riak.client.core.query.timeseries.Cell;
9  import com.basho.riak.client.core.query.timeseries.QueryResult;
10 import com.basho.riak.client.core.query.timeseries.Row;
11
```

```
12 import java.net.UnknownHostException;
13 import java.util.ArrayList;
14 import java.util.Arrays;
15 import java.util.Collections;
16 import java.util.List;
17 import java.util.concurrent.BrokenBarrierException;
18 import java.util.concurrent.CyclicBarrier;
19 import java.util.concurrent.ExecutionException;
20
21 public class RiakTsClientThread1 extends Thread {
22     private final String name = "";
23
24     public RiakTsClientThread1() {
25
26     }
27
28     @Override
29     public void run() {
30         try {
31             RiakClient client = RiakClient.newClient(8087,
    ↪ "localhost");
32             Query q = new Query.Builder(createTable1()).build();
33             QueryResult execute = client.execute(q);
34
```

```
35              for (int m = 1; m <= 1; m += 10) {
36                  System.out.print(m);
37                  for (int times = 0; times < 1; times++) {
38                      long start = System.currentTimeMillis();
39                      for (int i = 0; i < 20000; i++) {
40                          List<Row> list = new ArrayList<>(m);
41                          long time = System.currentTimeMillis();
42                          for (int j = 1; j <= m; j++) {
43                              list.add(new
    ↪ Row(Cell.newTimestamp(time), new Cell("sensor" + j), new
    ↪ Cell(Math.random() * 100)));
44                          }
45                          Store s = new
    ↪ Store.Builder("test21").withRows(list).build();
46                          client.execute(s);
47                      }
48
49                      System.out.print(";" +
    ↪ (System.currentTimeMillis() - start));
50                  }
51                  System.out.println("");
52              }
53
```

```
54          } catch (InterruptedException | UnknownHostException |
    ↪ ExecutionException e) {
55              e.printStackTrace();
56          }
57      }
58
59      private String createTable1() {
60          return "CREATE TABLE test1 ( time timestamp not null,
    ↪ sensor varchar not null, value double not null, Primary key
    ↪ ((sensor, quantum(time, 1, 'h')), sensor, time))";
61
62      }
63
64
65      private String dropTable1() {
66          return "Drop table test1";
67      }
68  }
```

E.2.2. RiakTsQueryThread

```
1  package at.tugraz;
2
3  import com.basho.riak.client.api.RiakClient;
```

```
4  import com.basho.riak.client.api.commands.timeseries.Query;
5  import com.basho.riak.client.core.query.timeseries.QueryResult;
6  import com.basho.riak.client.core.query.timeseries.Row;
7
8  import java.net.UnknownHostException;
9  import java.util.concurrent.ExecutionException;
10
11 public class RiakTsQueryThread extends Thread {
12
13     @Override
14     public void run() {
15         try {
16             RiakClient client = RiakClient.newClient( 8087,
   ↪ "localhost");
17
18             Query query = new Query.Builder("select count(*)
   ↪ from test1 where sensor = 'sensor1' and time > 1 and time <
   ↪ 9999999").build();
19             long start = System.currentTimeMillis();
20             QueryResult execute = client.execute(query);
21             System.out.println(System.currentTimeMillis() -
   ↪ start);
22
23         } catch (UnknownHostException e) {
```

```
24                 e.printStackTrace();
25             } catch (InterruptedException e) {
26                 e.printStackTrace();
27             } catch (ExecutionException e) {
28                 e.printStackTrace();
29             }
30         }
31 
32         public RiakTsQueryThread() {
33         }
34     }
```