DB Lasttest DBAP 11

15.01.2014, Henning Budde

Andreas Lockermann

Thomas Partsch

DB Lasttest

Das konzeptionelle Schema des FDBS wurde als Cloud-Datenbanksystem in Form einer Cassandra-Datenbank implementiert (siehe Anhang [A1]). Insgesamt sind momentan 69 Entity-Typen als Columnfamilies implementiert. Seit mehreren Monaten werden kontinuierlich Messdaten von verschiedenen Sensoren, die an die Teststrecke der FH Köln angekoppelt sind, in der Datenbank gespeichert. Die Summe der zurzeit erfassten Messwerte beläuft sich auf ca. 5 000 000. Es wurde beobachtet, dass auf eine Antwort von einfachen Web-Anfragen sehr lange gewartet werden muss. Aus diesem Grund wurde das globale Schema, das mittels des Cassandra DBMS umgesetzt worden ist, noch einmal analysiert und es wurden bestimmte Veränderungen vorgenommen, mit denen eine Steigerung der Performance und damit auch eine schnelle Antwortzeit bei Anfragen gewährleistet werden soll.

Dazu wird im Folgenden zunächst auf die Abbildung des Zeitstempels, dann auf die Darstellung von Messwerten und deren Bezeichnung und einen möglicher Performance-Gewinn durch den Einsatz von Messlinien eingegangen. In Abschnitt 2 wird auf die aktuelle Netzwerktopolgie vorgestellt, diese gewährleistet im Gegensatz zu der vorherigen eine bessere Performance.

1. Verbesserung der Performance durch Änderungen am Datenbankschema

Als Cloud-Datenbank wird Apache Cassandra verwendet. Das Schema dieser Datenbank ist gemäß dem Konzeptionellen Schemas (vgl. DBAP4) aufgebaut. Um unter Last eine akzeptable Performance zu haben, wird das Schema auf der Cassandra-Datenbank hinsichtlich bestimmter Attribute angepasst.

1.1 Zeitstempel in der Entität Messwert

Um Performance auf Seiten der Cloud-Datenbank Cassandra zu gewinnen, wird der Zeitstempel, der bisher als Long-Wert im Format eines Unix-Zeitstempels als Attribut *MesWerTimSta* vorgesehen ist, zerlegt und zusätzlich in die folgenden Attribute (Integer-Werte) gelegt:

MesWerTimYea (z.B. 2013)

MesWerTimMon (z.B. 9)

MesWerTimDay (z.B. 25)

MesWerTimHou (z.B. 14)

MesWerTimMin (z.B. 54)

MesWerTimSec (z.B. 30)

MesWerTimMs (z.B. 1000)

Der Vorteil von dieser Aufsplittung ist, dass bei SELECT-Anfragen konkret nach Daten gesucht werden kann, die z.B. im Jahr 2013 und im Monat 9 in der Datenbank hinterlegt worden sind. Gibt es nur ein Feld mit dem Unix-Zeitstempel, kann nur nach Wertebereichen gesucht werden (z.B. Zeitstempel > 1234567890123 AND Zeitstempel < 1334567890123). Diese Art der Suche erfordert die Operatoren > oder <, diese brauchen viel Performance, da die Datenbank dazu alle Datensätze durchlaufen muss. Eine konkrete Suche nach z.B. *MesWerTimYea* = 2013 scheint somit performanter und ist im Folgenden für eine Beispielanfrage dargestellt:

```
SELECT * FROM "Messwert" WHERE "MesWerSenID" = '8' AND "MesWerTimYea" = 2013 AND
"MesWerTimMon" = 12 AND "MesWerTimDay" = 4 AND "MesWerTimHou" = 13 AND "MesWerTimMin"
= 50  ALLOW FILTERING;
```

In diesem Beispiel werden alle Messwerte des Sensors mit der Sensor-ID „8“ ausgegeben, die zum Zeitpunkt 4.12.2013 um 13:50 Uhr in der Datenbank abgelegt worden sind.

1.2 Darstellung von Messwerten und ihrer Bezeichnung in der Entität Messwert

Die Daten von Sensoren, die mehrere Messwerte zu einem Zeitpunkt senden, werden laut dem konzeptionellen Schema als Messwert-Tupel in der Datenbank hinterlegt. Bei einem SELECT, bei dem man z.B. alle Werte erhalten möchte, die einen Temperaturwert von 30 Grad Celsius besitzen, müssen alle Daten selektiert und danach ausgewertet (gesplittet und nach Informationen des SensorProdukts interpretiert) werden.

Ein neuer Ansatz besteht darin, die Messwert-Tupel vor dem Einfügen in die Datenbank zu splitten, so dass die Werte einzeln in der Datenbank hinterlegt werden. Sendet ein Multiraumsensor z.B. Werte für Temperatur und Bewegung zu einem Zeitpunkt, so entsteht ein Datenbankeintrag der neben den Columns wie z.B. *MesWerID* zwei Columns Temperatur und Bewegung jeweils mit dem entsprechenden Messwert besitzt. Durch das flexible Schema der Cassandra-DB kann jeder Messwert-Datensatz dadurch flexibel nach seinen Werten aufgebaut werden. Bei einem SELECT ergibt sich der Vorteil, dass keine Interpretation der Messwerte notwendig ist, da der Column-Name die Information über den Messwert mitliefert. Zusätzlich benötigte Metadaten (wie z.B. die SensorSemantik) können aus dem Sensorprodukt entnommen werden.

Bezüglich der Umstrukturierung der Entität Messwert sieht die Darstellung wie folgt aus: Jeder Messwert, d.h. auch jeder Wert in einem Messwert-Tupel wird als einzelner Datensatz in der Datenbank abgelegt:

MesWer PRIK	**MesWer ID**	**MesWer SenID**	**MesWer PhyNam**	**MesWer Wer**	**MesWer TimSta**	**MesWer TimYea**	**...**
100	4711	2000	Temp	25	1380188454000	2013	...
101	4711	2000	Bew	50	1380188454000	2013	...
102	4711	2000	Qua	0	1380188454000	2013	...
103	4712	3000	Temp	27	1380188480100	2013	...
104	4712	3000	Bew	50	1380188480100	2013	...

Hier ist zu sehen, dass drei Messwerte eines 3er-Messwert-Tupels (gleiche SenID und gleicher Zeitstempel) als drei Datensätze in die Datenbank eingetragen werden. *Phynam* ist der physikalische Name (z.B. „temp“ für Temperatur) und *Wer* enthält den Messwert, z.B. „25“ für 25 Grad Celsius. Der Vorteil dieser Umstellung ist, dass kein Extrahieren der Messwerte für die Verwendung der Messwerte vorgenommen werden muss. Werte können direkt nach *Phynam* selektiert werden (z.B. alle Temperaturwerte). Nachteil ist, dass mehr Datensätze erzeugt werden (4er-Messwert-Tupel: 1 Datensatz vorher, jetzt 4 Datensätze). Dieses passt gut mit dem künftigen

Verschlüsselungskonzept für die Übertragung von Messdaten zusammen, wie es in den bereits definierten Protokolltypen der SensorCloud vorgesehen ist (vgl. [1]). Die darin vorgesehene Verschlüsselung von Messdaten verlangt eine singuläre Behandlung jedes Messwerts. Dabei braucht jeder zu verschlüsselte Datensatz bei dem im [1] gezeigten Verschlüsselungsverfahren eindeutig zugeordnete Parameter wie z.B. den Initialisierungsvektor (iv), der für die Initialisierung des Verschlüsselungsalgorithmus zur Steigerung der Konfusion genutzt wird.

1.3 Performance-Gewinn durch den Einsatz von MessLinien

Eine Messlinie enthält Messwerte eines Sensors, die über einen bestimmten Zeitraum erfasst worden sind. Demnach enthält diese Entität die Attribute MesLinID (UUID zur eindeutigen Identifikation einer MEssLinie), MesLinMesWerIDs enthält alle IDs (UUIDs), die zu einer MessLinie gehören. MesLinQueID gibt an, wozu die Messwerte dieser Messlinie gehören, MesLinTimBeg und MesLinTimEnd bestimmten einen Zeitraum. Die in diesem Zeitraum, bezogen auf einen Sensor (MesLinQueID), in der Datenbank hinterlegten Messwerte sind in dieser MessLinie in MesLinMesWerIDs hinterlegt. Des Weiteren gibt es das Attribut MesLinTyp für die Typisierung einer Messlinie, z.B. „S“ für Sensor und das Attribut MesLinAggFkt um zu wissen, mit welchen Mitteln (Funktionen) die erfassten Messwerte aggregiert werden dürfen.

Durch den Einsatz von Messlinien, d.h. es wird eine Messlinie pro Tag für einen Sensor erstellt, die in dem Attribut MesLinMesWerIDs die IDs der zugehörigen Messwerte hält, soll eine bessere Performance gewährleistet werden. Der Grund dafür ist wie folgt zu verstehen: Durch eine SELECT-Anfrage an MessLinie erhält man alle IDs, die zu einem Tag zu einem Sensor gehören. Dies bedeutet, wenn für eine Visualisierung alle Messwerte eines Tages für einen Sensor gefragt sind, werden mit dieser SELECT-Anfrage alle benötigten IDs aus der Datenbank geholt. In einem zweiten Schritt wird zu jeder dieser IDs der entsprechende Messwert aus der Datenbank geladen und es kann z.B. eine Darstellung/Visualisierung der Messwerte erfolgen. Durch den Einsatz von MessLinie muss demnach nicht erst anhand des Zeitstempels nach Messwerten gefiltert werden, die in einem Zeitraum (z.B. einem Tag) von einem Sensor erzeugt worden sind, da dies durch zyklische Vorarbeit in der Entität Messwert hinterlegt und jederzeit abgefragt werden kann.

1.4 Performance-Untersuchung

Im Folgenden werden die in den vorangehenden Kapiteln diskutierten Änderungen in ihrer Zugriffsdauer verglichen. Für die Untersuchung wird als Anwendungsfall die Generierung eines Tages-Diagramms mit stündlichen Durchschnittsmesswerten angenommen. Hierbei können die dafür notwendigen Messwerte auf verschiedene Art selektiert werden.

Die Analyse fand auf einem virtualisierten Cassandra-Cluster in der Version 1.1.x mit zwei Knoten statt. Die Menge der Messwerte in der Tabelle lag zu dem Messzeitpunkt bei ca. 400.000 Messwerten. Die Menge der gesuchten Messwerte des Tages lag bei 17.400 Messwerten.

Zugriffsart A in Abbildung 1 illustriert die Dauer eines Selects mit einschränkendem Timestamp-Intervall, um alle Datensätze eines Tages zu selektieren. Dieser Zugriff dauerte mit 240 Sekunden am längsten, da über die Angabe eines Intervalls nicht direkt auf den Key-Value-Store der Cassandra zugegriffen werden kann. Werte können nicht direkt über den Key adressiert werden und müssen durch sequentielles Lesen des gesamten Datenbestandes und direktem Vergleich von dem DBS ermittelt werden.

Zugriffsart B dauerte 62 Sekunden. Hierbei wurde der Timestamp in Jahr, Monat und Tage aufgeteilt und darüber Secondary Indizes gelegt. Bei dieser Zugriffsart können die Datensätze direkt aus dem Key-Value-Store der Cassandra-Datenbank gelesen werden.

Zugriffsart C spiegelt den Zugriff auf die kompletten 17.400 Messwerte des Tages über den Primary-Key wieder. Der Zugriff dauerte 13 Sekunden. Es zeigt sich hierbei, dass ein Direktzugriff über den Primary-Key schneller ist, als der Zugriff über mehrere Secondary-Keys.

Zugriffsart D dauerte 4,3 Sekunden. Hierbei wurde der Timestamp wie bei der Zugriffsart C aufgeteilt und zusätzlich die Stunde mit aufgenommen. Es wurden 24 Selects durchgeführt und zu jeder Stunde mit der Angabe von „limit 1“ auf den ersten gefundenen Datensatz beschränkt. Dieser Zugriff erscheint sinnvoll, wenn man aus einer Menge von Messwerten einen Messwert pro Stunde als repräsentativ ansieht.

Zugriffsart E dauerte 0,02 Sekunden. Hierbei wurde auf die Menge der in Zugriffsart D selektierten Datensätze direkt über deren Primary-Key zugegriffen.

Die Untersuchung zeigt, dass der Einsatz von Messlinien und eine Aggregierung von Messwerten im Batch-Betrieb einen deutlichen Performancegewinn bedeutet. Messwerte könnten periodisch kumuliert werden und als neue Messwerte in der Messwerttabelle abgespeichert werden. Deren Keys würden dann in einer Messline hinterlegt werden. Der Zugriff auf 24 vorher schon berechnete Durchschnittswerte und Zugriff auf diese Werte über deren Primary-Key um den Faktor 12.000 schneller als über den Zugriff auf alle Messwerte des Tages über ein Timestampintervall.

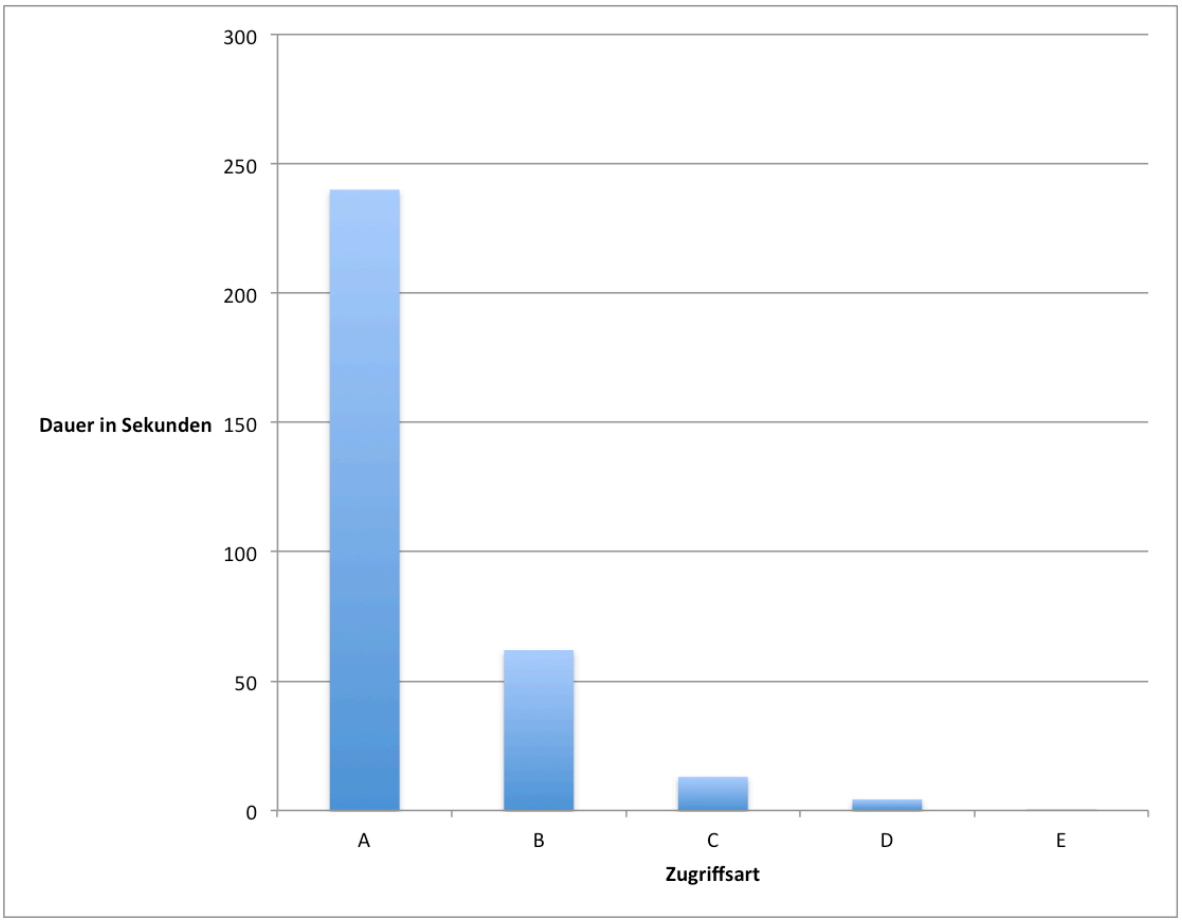

Abbildung 1: Performance-Messung verschiedener Zugriffsarten

2. Verbesserung der Performance durch Skalierung von Rechnerknoten

Die folgende Abbildung zeigt die Netzwerktopolgie SensorCloud an der FH Köln. Die Datenbank Apache Cassandra ist mit ihren Knoten (Cassandra Nodes) dargestellt. Das Hinzufügen von weiteren Cassandra-Knoten ist möglich. Zudem ist ein Clustering von Openfire möglich. Ein vor die Tomcat- und Apache-Server geschalteter IP-Load-Balancer sorgt dafür, dass die Last auf die verschiedenen Server verteilt wird. Durch diese Skalierungsmöglichkeiten soll eine gute Performance erreicht werden.

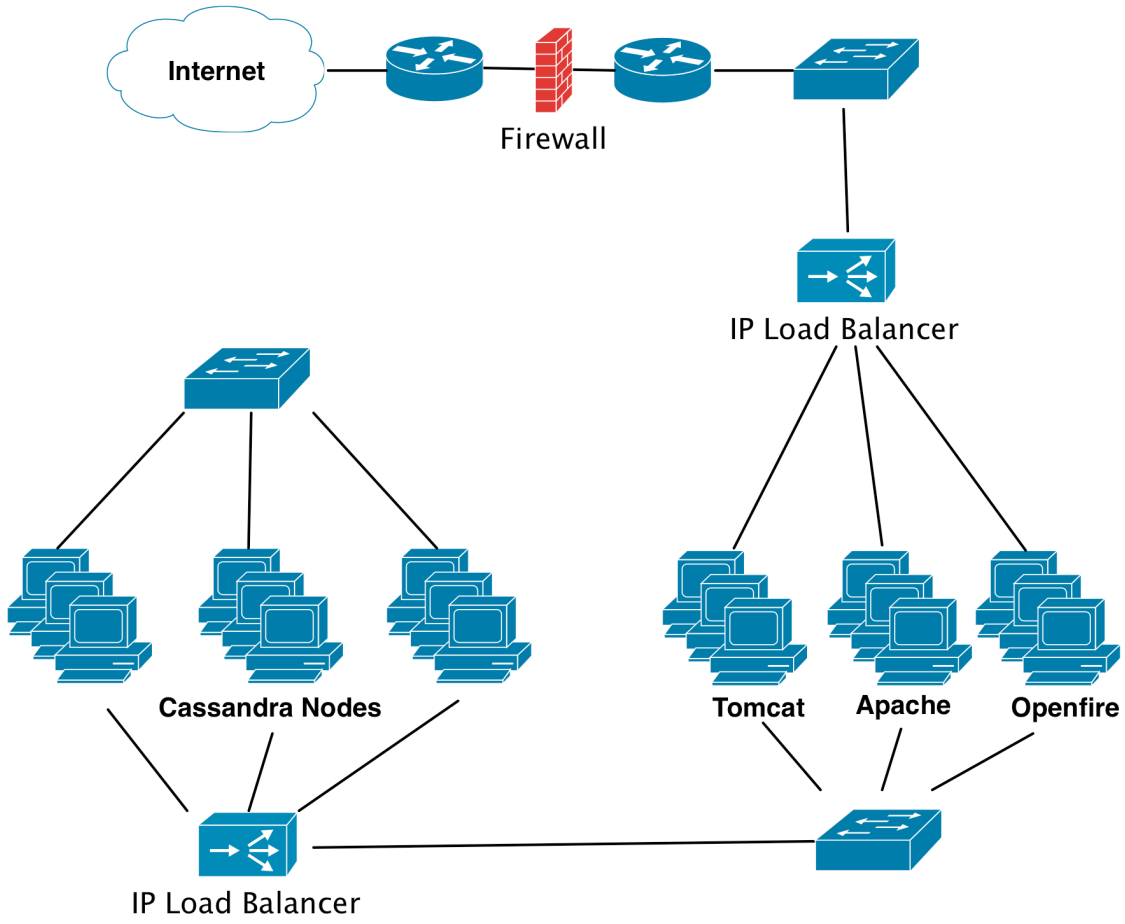

Abbildung 2: Netzwerktopologie SensorCloud FH Köln

2.1 Performance Analyse mit Cassandra Nodetool

Um die Performance eines Cassandra Cluster zu messen bringt das DBMS eine eigenständige Werkzeugsammlung namens Nodetool mit. Das Nodetool vereinigt Programme mit Hilfe derer sich Cassandra analysieren lässt. Für einen Performance Vergleich steht das Tool cfhistograms (**c**olumn **f**amily **h**istograms) zur Verfügung.

Dieses Tool lässt sich für jede Columnfamily separat starten.

```
nodetool cfhistograms <Keyspace_Name> <ColumnFamily_Name>
```

Die hier gezeigten Werte wurden vor einer Performance Optimierung durchgeführt um spätere Performance Verbesserungen messen zu können.

Auszug cfhistograms der Column Family „Messwert“

Offset	SSTables	Write Latency (micros)	Read Latency (micros)	Partition Size (bytes)	Cell Count
1	1012190	0	0	0	0
2	1776	0	0	0	13
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0
6	0	0	0	0	0
7	0	0	0	0	0
8	0	0	0	0	0
10	0	0	0	0	0
12	0	0	0	0	2883034
14	0	0	25	0	0
17	0	0	3056	0	0
20	0	0	6719	0	0
24	0	0	13133	0	0
29	0	0	29476	0	0
35	0	0	18246	0	0
42	0	0	14571	0	0
50	0	0	19097	0	0
60	0	0	8322	0	0
72	0	0	4214	0	0
86	0	0	2903	0	0
103	0	0	1911	0	0
124	0	0	1624	5	0
149	0	0	1057	8	0
179	0	0	407	0	0
215	0	0	1837	0	0
258	0	19	412597	0	0
310	0	69	253519	0	0
372	0	725	81303	0	0
446	0	4209	29026	0	0
535	0	18455	12102	2878908	0
642	0	22716	26754	0	0
770	0	17022	20907	0	0
924	0	15622	9969	4126	0
1109	0	22789	4100	0	0
1331	0	54219	1948	0	0
1597	0	46827	1470	0	0
1916	0	25189	685	0	0
2299	0	3075	737	0	0
2759	0	1159	718	0	0
3311	0	755	384	0	0
3973	0	492	512	0	0
4768	0	207	279	0	0
5722	0	70	289	0	0

Erläuterung:

Offset
Die Spalte Offset ist gültig für jede andere Spalte und gibt je nach ausgewählter zusätzlicher Spalte eine andere Maßeinheit wieder.

SSTables

Verteilung der Daten auf SSTables einer Cassandra Datenbank. Offset gibt die Anzahl der SSTables an.

Beispiel:

Offset	SSTables
1	1012190
2	1776

1012190 Datensätze sind in einem SSTable und 1776 Datensätze sind in zwei SSTables.

Write Latency

Offset gibt die Schreibgeschwindigkeit in Mikrosekunden (0,000 001 Sekunden) an.

Beispiel:

Offset	Write Latency
372	725
446	4029

725 Datensätze benötigten 372 Mikrosekunden (pro Datensatz) um geschrieben zu werden. 4029 Datensätze benötigten jeweils 446 Mikrosekunden um geschrieben zu werden.

Read Latency

Offset gibt Lesegeschwindigkeit in Mikrosekunden (0,000 001 Sekunden) an.

Beispiel:

Offset	Read Latency
372	81303
446	29026

81303 Datensätze benötigten 372 Mikrosekunden (pro Datensatz) um gelesen zu werden. 29026 Datensätze benötigten jeweils 446 Mikrosekunden um gelesen zu werden.

Partition Size

Offset gibt Größe in Byte pro Datensatz an.

Beispiel:

Offset	Partition Size
535	2878908

2878908 Datensätze sind jeweils 535 Bytes groß.

Cell Count

Offset gibt Anzahl Spalten (Columns) eines Datensatzes an.

Beispiel:

Offset	Cell Count
12	2883034

2883034 Datensätze haben jeweils 12 Spalten.

Literatur

[1] Protokolltypen der SensorCloud: „SensorCloud JSON Representation“ (Draft) der Protokoll-AG des Konsortium SensorCloud, Version 0.2, Stand: September 2013.