**Analyse Datenbankanforderungen**

Ziel des Projekts bezüglich der verwendeten Datenbank sind die Weiterverwendung des bestehenden Datenmodells und die Recherche nach möglichen Alternativen.

Für die Weiterverwendung des bestehenden Modells spricht, dass es bereits mehrere Gruppen gibt, die dieses System verwenden [Oberluggauer], [SmartVis]. Für die Ablösung des bestehenden Modells spricht eine eventuell bessere Performance alternativer Datenbanken bei großen Datenmengen.

Da die Weiterverwendung gewünscht wird, werden zuerst Performancemessungen auf einem Relationalen Datenbanksystem durchgeführt. Ist die Performance ausreichend, wird diese Datenbank verwendet.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Eigene Vorkenntnisse | Popularität | Open Source (Kostenlos) |
| MySQL | 3 | 3 | 1 |
| PostreSQL | 1 | 2 | 1 |
| MariaDB | 0 | 1 | 1 |

Tabelle 1: Auswahlkriterien Relationale Datenbank

Für die Auswahl der Datenbank wurden die Vorkenntnisse und freie Verfügbarkeit als Kriterien definiert. Da die Erfahrung mit Datenbanken in der Projektgruppe allgemein eher gering ist, ist die erwartete Unterstützung aus der Community ein weiteres Kriterium. Dafür wurde die Anzahl der Fragen innerhalb einer Woche auf Stack Overflow verwendet. Nach den Kriterien aus (Tabelle1) ist die Wahl auf MySQL gefallen.

Zum Zeitpunkt der Messung sind wir von ungefähr 250.000.000 Datensätzen innerhalb eines Jahres ausgegangen. Das ergibt sich aus 500.000 Smartmetern der Salzburg AG (1 Wert pro Tag) und ungefähr 60.000.000 Messwerten aus den REDD Daten.

**Testdaten**

Als Testdaten wurden die ‚low\_freq‘ Daten aus dem REDD Datensatz verwendet. Die Daten liegen in pro Haus in einem eigenen Ordner und dort pro Kanal in einem eigenen File. Die Files sind ‚channel\_X.dat‘ benannt wobei X eine fortlaufende Nummer ist. Parallel zu den Messwerten liegt eine Datei ‚labels.dat‘ in welche die Zuordnung der einzelnen Kanäle zu den Verbrauchern ermöglicht. Die Messwerte sind wie in (Listing1) dargestellt in einem CSV Format abgespeichert, wobei das Trennzeichen ein Leerzeichen ist.

Listing 1: CSV Format der REDD Daten

timestamp\_power

**Messung**

Für die Performance Messungen wurde eine C# Applikation geschrieben, welche sämtliche ‚low\_freq‘ REDD Daten in eine MySQL Datenbank importiert. Dafür wurden die Datensätze pro Kanal geparst und in 100.000er Schritten in die Datenbank importiert. Aus dem Zeitstempel im Unix-Epoch-Format wurde lokal der Tag und der Monat ausgerechnet um später danach gruppieren zu können. Zusätzlich wurde pro Kanal eine fortlaufende Id, in (Bild1) als ‚meterId‘ dargestellt, vergeben.



Abbildung : Tabelle REDD mit Testdaten

Damit das Einfügen von 100.000 Datensätzen auf einmal funktioniert wurde die maximal erlaubte Paketgröße auf 160 MB erhöht [https://dev.mysql.com/doc/refman/5.7/en/packet-too-large.html]. Die Datenbank läuft auf dem lokalen Testsystem (i5 4690K @ 3.5GHz, 16GB Arbeitsspeicher, Windows 10 Pro) um einen möglichen Delay über das Netzwerk ausschließen zu können.

Nach dem Hochladen eines jeden Datensatzes wurde der Durchschnittsverbrauch pro Id, Monat und Tag abgefragt (Listing 2).

**select** **SQL\_NO\_CACHE** **avg**(**power**) **as** **power**, **day**, **month**, meterId

**from** redd

**group** **by** meterId **asc**, **month** **asc**, **day** **asc**;

Listing 2: Berechnung des Durchschnittsverbrauchs pro Meter, Tag und Monat

Durch die SQL\_NO\_CACHE Anweisung wird verhindert das das Ergebnis der Abfragen aus dem Cache zurückgeliefert werden, was einem realistischen Szenario entspricht. Die Abfrage wurde fünf Mal wiederholt ausgeführt und die jeweilige Zeit mittels der ‚System.Diagnostics.Stopwatch‘ Klasse gemessen. Die Messungen wurden in einem XML Dokument abgespeichert um dann mit Excel weiterverarbeitet werden zu können. Aus den 5 Messungen wurde der Median und Mittelwert berechnet, welche dieselben Schwankungen zeigen. Daher wurde auf die Darstellung des Mittelwerts verzichtet.

Abbildung : Dauer Berechnung des Mittelwerts auf der MySQL Datenbank

(Abbildung 2) zeigt, dass die Dauer der Berechnung des Durchschnittsverbrauches in linearem Zusammenhang zur Anzahl der Datensätze in der Tabelle ist. Die teilweise starken Schwankungen des Medianes lassen sich durch Hintergrundprozesse wie Virenscanner und verschiedene Updatedienste erklären. Die Formel der Regressionsgerade liefert im Gegensatz zur y-Achse eine erwartete Dauer, bei der Abfrage auf X Datensätzen, in Millisekunden.

Abbildung 3: Dauer des Einfügens in die MySQL Datenbank

Auch das Einfügen neuer Datensätze steht, wie in (Abbildung 3) zu sehen in linearem Zusammenhang mit der Anzahl der neu eingefügten Datensätze. Die Tabelle mit den Messungen ist im Anhang zu finden.

**Erwartete Datenmengen**

Die Datenbank soll eine Sammlung von Messdaten aus verschiedenen Quellen sein. Im Moment sind vier verschiedene Datenquellen bekannt (Tabelle 1).

|  |  |  |
| --- | --- | --- |
| **Name** | **Anzahl** | **Kommentar** |
| REDD | 56M | bisher nur low\_freq |
| ADRES | 36M | 30 Haushalte je 1 Sensor, 2 Wochen, 1Hz |
| GREEND | 2.270M | 8 Häuser je 9 Sensoren, 1 Jahr, 1Hz |
| Salzburg AG | 182M / Jahr | 500.000 Haushalte 1 Messung pro Tag |

Tabelle : Erwartete Anzahl an Datensätzen

Diese Datensätze zusammen gerechnet beinhalten ungefähr 2.544 M Einträge. Rechnerisch würde eine Abfrage des Durchschnittswertes auf dem Testsystem ungefähr 40 Minuten dauern. Da moderne Datenbankserver allerdings um ein vielfaches performanter als ein Heimcomputer sind, ist dieser Wert für ein Livesystem nicht aussagekräftig. Es zeigt sich ebenfalls dass der derzeitig verfügbare Testserver (Xeon E5-2620 @ 2GHz, 4GB Arbeitsspeicher), für einen Betrieb mit allen Datensätzen nicht geeignet ist. Mögliche Ansätze zur Erhöhung der Performance wären horizontale Skalierung mit einem SQL System oder vertikale Skalierung mit einem NoSQL System. Eine Lösung dieses Problems ist für das nächste Semester geplant.