Lessons Learned - Cassandra

Lessons learned aus der Installation eines Cassandra-Setups mit 3 Nodes mithilfe Docker Compose, um ein soziales Netzwerk zu betreiben, das Daten von Twitter lädt.

Infrastruktur

- Cassandra ist sehr rechenintensiv. Auf dem PC, auf dem wir die Abgabe ausgeführt haben, kam es zwischenzeitig zu einem Absturz, da der PC überhitzte.
- Ein einzelner Cassandra-Docker-Container braucht lange zum Starten und Laden der Konfiguration (abhängig vom Hostsystem). Um sicherzustellen, dass es überall ohne Absturz läuft, haben wir einen „health check“ in Docker Compose implementiert. Dies war nicht einfach, da wir die richtigen Parameter durch Trial-and-Error ermitteln mussten. Zusätzlich haben die Container eine „restart always“-Policy, falls sie abstürzen.

```
healthcheck:
        test: ["CMD", "cqlsh", "-e", "describe keyspaces" ]
        interval: 10s
        timeout: 10s
        start_period: 50s
        retries: 10
```

- Cassandra bietet keine Weboberfläche, aber glücklicherweise hat die Open-Source-Community eine Weboberfläche namens cassandra-web implementiert. Cassandra-web erfordert, dass IPs statisch gesetzt werden, da es versucht, sich mit ihnen zu verbinden und nicht mit Docker DNS funktioniert.
- Cassandra-web erfordert eine ältere Ruby-Version (eine Version > 3 verursacht Probleme).
- Cassandra erfordert eine bestimmte Konfiguration, um zu funktionieren. Diese Konfiguration kann über Umgebungsvariablen und Konfigurationsdateien eingestellt werden. Die folgenden Informationen sind erforderlich und können als Umgebungsvariablen in Docker Compose gesetzt werden:
    - CASSANDRA_SEEDS: "cass1,cass2"
    - CASSANDRA_CLUSTER_NAME: SolarSystem
    - CASSANDRA_DC: Mars
    - CASSANDRA_RACK: West
    - CASSANDRA_ENDPOINT_SNITCH: GossipingPropertyFileSnitch
    - CASSANDRA_NUM_TOKENS: 128

- In unserem Fall haben wir zusätzlich die folgenden Parameter gesetzt, um das System nicht zu überlasten:
    - MAX_HEAP_SIZE: 1024M
    - HEAP_NEWSIZE: 1024M
- Erweiterte Konfigurationen können in den folgenden erforderlichen Konfigurationsdateien (gespeichert in etc/cassandra) gesetzt werden:
    - cassandra-env.sh
    - cassandra-rackdc.properties
    - cassandra.yaml
    - commitlog_archiving.properties
    - jvm-clients.options

    - jvm-server.options
    - jvm8-clients.options
    - jvm8-server.options
    - jvm11-clients.options
    - jvm11-server.options
    - logback.xml
- Parameter, die wir zusätzlich in diesen Dateien setzen mussten, waren:
    - `enable_materialized_views: true`, um materialisierte Ansichten erstellen zu können.
    - `enable_sasi_indexes: true`, um Indizes auf twitter.tweets-Inhalten zu erstellen.
    - `*_timeout: >default`, um große CSV-Dateien in das Cluster laden zu können.
- Das Docker-Volume, das auf den Cassandra-Container bei etc/cassandra gemappt ist, muss alle oben genannten Dateien enthalten, damit Cassandra funktioniert. Die in den oben genannten Umgebungsvariablen angegebenen Werte überschreiben die Standardwerte in den eigentlichen Konfigurationsdateien.
- Um die Einstellungen für cqlsh (ein Python-Skript-Shell zum Ausführen von CQL-Abfragen auf der Datenbank) zu setzen, haben wir eine csqhrc-Datei in /.cassandra hinzugefügt, die nur eine Datei im Verzeichnis mit den Einstellungen für cql erlaubt. Dort haben wir folgende Einstellungen geändert:

```
[ui]
timezone = Etc/UTC` : Set the current timezone
time_format = %d/%m/%Y %H:%M`: Change the date pattern to import
time_stamps
[copy]
ESCAPE = \: set the escape character
QUOTE = ": set the quote character
```

- Beim ersten Versuch hatten wir Probleme beim Ausführen unseres Startskripts zum Ausführen von CQL-Befehlen:

```
Connection error: ('Unable to connect to any servers',
{'172.20.0.6:9042': ConnectionRefusedError(111, "Tried connecting to
[('172.20.0.6', 9042)]. Last error: Connection refused")}).
```

  Dies lag an der aktivierten Authentifizierung, die wir dann für die Verbindung verwendet haben. Der Standardbenutzer und das Standardpasswort sind *cassandra*.

- Beim Erstellen von Materialized Views erhielten wir die folgende Meldung:

```
Warnings : Materialized views are experimental and are not
recommended for production use.
```

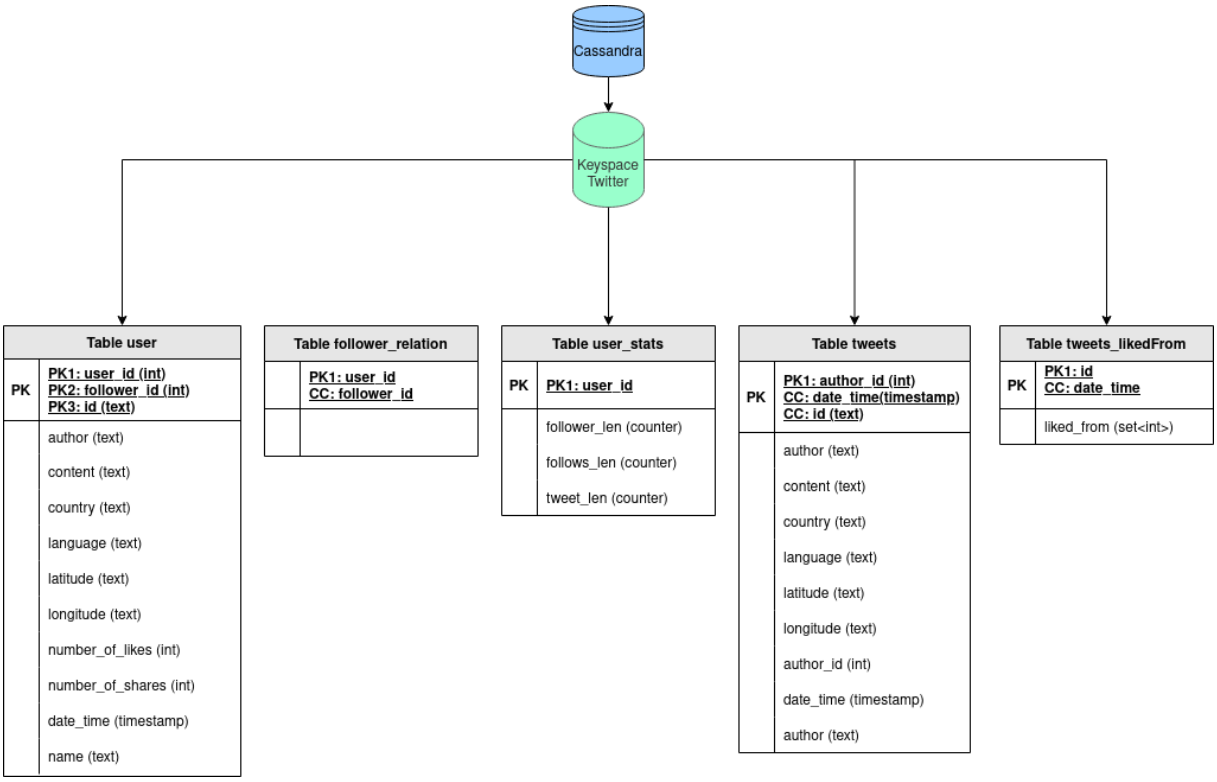
  Offenbar sind Materialized Views experimentell, und deshalb mussten wir sie explizit in der Konfiguration aktivieren. Es wird empfohlen, stattdessen doppelte Tabellen zu verwenden, wie in diesem Stackoverflow Thread vorgeschlagen.

- Views sind ebenfalls experimentell und müssen in der cassandra.yml-Datei aktiviert werden. Sie sind jedoch nur nützlich, um die Reihenfolge der „keys“-Spalte neu anzuordnen oder eine neue „no key“-Spalte als neuen Schlüsselwert hinzuzufügen. Es ist nicht möglich, neue Felder wie count(col_name) zu einer Ansicht hinzuzufügen.

Um dies zu erreichen, wird empfohlen, ein neues Datenschema zu verwenden und Daten mit dem cassandra-spark-connector als neue Tabelle zu laden.

- Um große CSV-Dateien zu importieren, wird empfohlen, den stableloader oder einen Spark-Cluster zu verwenden. Aufgrund der Zeitbeschränkung haben wir nur versucht, fertige cvs_to_sstable_convert-Tools zu verwenden und nicht versucht, sie manuell zu konvertieren. Die Daten konnten jedoch nicht in die Datenbank importiert werden, und wir erhielten eine korrekte Datei-Upload-Meldung mit *0-files upload*.
- Da wir nicht in der Lage waren, den stableloader zum Importieren großer Dateien zu verwenden, mussten wir die *liked_from_user*-Liste für jeden Tweet minimieren. Wir reduzierten die Daten von den ursprünglichen Likes (etwa 7 GB und neue Benutzer-IDs wurden generiert) auf ein Zehntel der Größe (etwa 480 MB).

Datenmodell



- Als wir zum ersten Mal die Daten in Cassandra importiert haben, wurde folgender Fehler geworfen:

  ```
  Failed to import 1 rows: ParseError - Failed to parse 5.34896E+17 :
  invalid literal for int() with base 10: '5.34896E+17', given up
  without retries 'builtin_function_or_method' object has no attribute
  'error'.
  ```

  Um dies zu beseitigen, mussten wir die Daten manipulieren, bevor wir sie manuell importiert haben.

- Wir mussten das Datenschema mehrmals ändern und verschiedene Kombinationen ausprobieren, um die Abfragen zum Laufen zu bringen. Schließlich haben wir die Relationstabelle mit den Beziehungen zwischen *user_id*, *follower_id* und *tweet_id* verwendet, die den Primärschlüssel bildet. Dadurch werden die Daten mehrmals für jede ID gespeichert und sie sind etwa 45x größer als die Originaldaten. Zusätzlich

verwenden wir eine Statistik-Tabelle mit Zählern, um schnell Abfragen zur Länge von Followern oder Follows durchführen zu können. Aufgrund der "world-search"-Abfrage (Aufgabe 4.5) haben wir auch die Tweets in einer separaten Tabelle hinzugefügt, um einen Index auf der Inhalts-Spalte zu erstellen und mit dem LIKE-Schlüsselwort filtern zu können.

- Benutzerdefinierte Typen (UDTs): Wir haben versucht, die Tweets als UDTs in das obige Datenschema zu laden. Um dies durchzuführen, haben wir die Struktur der kombinierten CSV-Datei anhand eines Stackoverflow Beitrags aktualisiert, erhielten aber immer einen Fehler für Spaltenungleichheiten. Wir vermuten, dass der Inhalt in den UDTs nicht in Anführungszeichen ist, was zu Fehlern mit „„ führte.
- SASI-Indizes sind (wie viele Dinge) experimentell und werden nicht für den Produktionsgebrauch empfohlen. Wir mussten sie jedoch für Übung 4.5 verwenden, um die Suche mit der LIKE-Anweisung zu ermöglichen. Wir nehmen an, dass Cassandra eine interne Tabelle mit jedem Wort erstellt, das mit den Tweets verknüpft ist, in denen es erscheint, da der Index etwa 5 Minuten zur Erstellung benötigt.

Weiteres

- Um für Lesen über Schreiben zu optimieren (da dies unser Fall sein wird), haben wir die Compaction auf *LeveledCompactionStrategy* eingestellt, wie es von der Cassandra-Dokumentation für diese Art von System empfohlen wird:

  ```
  [..] WITH compaction = {'class' : 'LeveledCompactionStrategy'};
  ```
- Formel für den Replikationsfaktor, wie von diesem Blog Beitrag vorgeschlagen:
  [read-consistency-level] + [write-consistency-level] > [replication-factor]
- Vorsortierung der Daten kann durchgeführt werden durch:
  `CLUSTERING ORDER BY (number_of_likes ASC);`