Assignment 1

Uploading Data to the Database

Database Tuning

Gruppe K
Brezovic Ivica, 11702570
Demir Cansu, 11700525
Mrazovic Mirna, 11700383

March 15, 2020

Experimental Setup

Das Setup ermöglicht es, die Zeit zu vergleichen, die der Import von tsv-Dateien in ein Postgres-Datenbankschema mit unterschiedlichen Ansätzen benötigt.

Die Eingabedaten werden in zwei Dateien bereitgestellt. Die erste Datei, 'auth.tsv', enthält zwei Spalten mit Zeichenfolgen und besteht aus ca. 3 Mio Zeilen. Die zweite Datei, 'publ.tsv', hat 6 Spalten mit Zeichenfolgen und ca. 1 Mio Zeilen.

Die Daten werden aus den Dateien gelesen und mit Hilfe einer von zwei Approaches in die Datenbank hochgeladen. Für jeden Ansatz wird die Zeit für die Durchführung der eigentlichen Einfügungen gemessen, indem vom Start-Zeitstempel nanoTime() die Zeit subtrahiert wird, wenn alle Zeilen eingefügt worden sind.

Die Zeit für die Herstellung der Verbindung zur Datenbank am Anfang und für das Schließen der Datenbank beim Beenden des Programms wird nicht berücksichtigt.

Straightforward Implementation

Implementation Die .tsv-Dateien werden in einen Java FileStream und den zugehörigen Scanner eingelesen - aufgrund der Dateigröße. Die Dateien werden dann Zeile für Zeile geparst, wobei jede Zeile am Tabulator-Trennzeichen in ihre Tupel aufgeteilt wird. Für jeden vollständigen Satz von Spalten (eine Zeile) wird dann eine einfache INSERT-Anweisung ausgegeben:

```
INSERT INTO \"" + "Auth" + "\" VALUES (" + valstr + ")
```

Der Grund, warum dieser Ansatz sehr langsam ist, liegt darin, dass für jede Zeile in der .tsv-Datei (d.h. für jedes Tupel) eine neue Anfrage an den Server gesendet werden muss. Wenn der Server die Anfrage erhält, muss er sie parsen, optimieren und die Auswahl des Zugriffspfades berechnen, was einen Overhead (Netzwerklatenz und Berechnungen) erzeugt. Auch aufgrund der Ausführung jeder einzelnen Übertragung als eine einzelne Transaktion entsteht ein erheblicher Übertragungs-Overhead.

Efficient Approaches

Efficient Approach: PostgreSQL COPY von stdin

Implementation Der Befehl COPY kopiert Daten zwischen PostgreSQL-Tabellen und Dateien. Alle Zeilen einer Datei konnen dadurch mit einem einzigen Befehl in eine Tabelle geladen werden, anstatt mehrere INSERT-Befehle dafür zu verwenden. Außerdem ist der COPY-Befehl dafur optimiert, bulk loads durchzufuhren, also eine große Anzahl an Zeilen zu laden. Dadurch ist er aber auch weniger flexibel als ein INSERT-Befehl.

Der JDBC CopyManager wird verwendet, um all dies für uns zu handhaben und wird mit einem FileStream dargestellt. Intern empfängt der Server den Inhalt der Eingabedatei direkt als Stream und interpretiert das Tabulator- und Newline-getrennte Format.

```
COPY \"" + "Auth" + "\" FROM stdin", new FileInputStream(fileName),178
```

Why is this approach efficient? Das System muss nicht viele Abfragen parsen, optimieren und ausführen, sondern kann stattdessen Daten so schnell akzeptieren, wie das Datenbank Backend dazu in der Lage ist (das Parsen der Trennzeichen verursacht weitaus weniger Overhead als die Abfrageausführung). Indexes können auf einmal aktualisiert werden, wenn alle Daten empfangen wurden.

Tuning principle Der Ansatz folgt dem Tuning-Prinzip "Render on the server what is due on the Server", da beim Copy-Befehl die zu lesende oder zu schreibende Datei auf dem Database Server Rechner, nicht auf dem Client, liegen oder für diesen zugänglich sein muss (oder sie kann von stdin gelesen werden, wodurch der Empfang eines entfernten Datenstroms effektiv ermöglicht wird). Die Dateien müssen dem PostgreSQL-Benutzer (über stdin) zugänglich gemacht werden oder für ihn lesbar oder schreibbar sein (die User ID, als die der Server läuft), nicht für den Client. [**1**]

Darüber hinaus grenzt es auch an das bereits erwähnte Prinzip "Start-up costs are high; Running costs are low", weil das COPY effektiv zu einem großen Commit führt, sobald alle Daten gelesen wurden, wodurch der COMMIT-Overhead eliminiert wird.

Portability

Implementation Wichtig ist es, die Connection zur Datenbank herzustellen. Das Einzige, was zu tun ist, ist den nicht gewünschten Block auszukommentieren. Beispielsweise ist im folgenden Codeblock der Server 'db tuning ss2020' auskommentiert. Das Programm wird auf dem localhost ausgeführt. Die Daten dienen zum Zugriff zu der Datenbank. Diese muss jeder individuell für seine PostgreSQL Datenbank anpassen. x

```
----- db tuning ss2020 -----
/*String host = "biber.cosy.sbg.ac.at";
String port = "5432";
String database = "dbtuning_ss2020";
String pwd = "**********";
String user = "cdemir";*/
--------- localhost ---------
String host = "localhost";
String port = "5432";
String database = "postgres";
```

```
String pwd = "*******";
String user = "postgres";
```

Did you observe performance differences. If so: Why. If not: Why not? Der Grund weshalb bei Localhost eine deutlich geringere Runtime ausgegeben wird als bei der Ausführung auf dem Server, ist folgender: Localhost befindet sich auf meinem Rechner - somit fällt die Übertragungszeit weg.

Runtime Experiment

Notes

- For the straightforward approach you are allowed to import only a subset of the tuples (e.g., 10.000 tuples) and estimate the overall runtime. The timings for all other approaches should be real measurements over the whole data set.
- Specify the setting of the experiment, i.e., where is the database server (local machine, database server at the department), where is the client (wired/wireless network of the department)?

Setting of the Experiment

- Für das Experiment ist die Implementierung so eingerichtet, dass auth-small.tsv (mit 10.000 Tuples) in ein PostgreSQL-Server-Schema geladen wird. Die Zeit von kurz vor dem Lesen/Einfügen der ersten Zeile bis zum Abschluss des Einfügens wird protokolliert.
- Aus der Laufzeit von 10.000 Tupeln kann dann die Gesamtzeit errechnet werden.
- Setup is a local PostgreSQL server (Version: 12.2) and client on an MacBook Pro with following hardware: i5 - 2,3 GHz - 128 GB SSD and 8GB memory.
- Datenbankserver: biber.cosy.sbg.ac.at des Fachbereichs
- Notes to (1): Number of rows: 5000, Speed: 156.19008 rows/s
- Notes to (2): Number of rows: 3095201, Speed: 37730.152 rows/s
- Notes to (3): Number of rows: 3095201, Speed: 562.5229 rows/s
- Notes to (4): Number of rows: 3095201, Speed: 1124197.8 rows/s

PostgreSQL	Runtime [sec]
(1) Straightforward	32.01227
(2) Approach (COPY)	82.03521
Localhost	Runtime [sec]
(3) Straightforward	8.88852
(4) Approach (COPY)	2.75325

Time Spent on this Assignment

Time in hours per person: **8**

References

[1] https://www.postgresql.org/docs/current/sql-copy.html