Projektaufgabe 2

Effiziente Matrixmultiplikation (5 Punkte) + 2 Bonus Punkte

In der zweiten Projektaufgabe untersuchen wir, wie effizient Matrixmultiplikationen innerhalb Ihres DBMS ausgeführt werden können. Dabei vergleichen wir 2 Ansätze innerhalb des DBMS mit einem, den Sie in der Programmiersprache ihrer Wahl implementieren. Die Matrizen werden zum Teil wieder sparse sein. Die konkreten Ansätze zur Matrixmultiplikation werden nicht in der Vorlesung besprochen, sondern Sie entnehmen sie dem beigefügtem Lehrbuchausschnitt aus „Data Management in Machine Learning Systems“ aus Beispiel 2.1 und 2.2.

Hinweis: Es ist beabsichtigt, dass Sie viele Ihrer Konzepte und Implementierungen aus der ersten Projektaufgabe übernehmen können. Der Aufbau der Phasen ist daher ähnlich.

Phase 1 – Legen der Basis (2.5 P)

In der ersten Phase legen Sie die Basis für das Gesamtprojekt. Dies umfasst:

- **Datengenerator für Matrizen mit Sparsity (0.5 Punkte):** Realisieren Sie eine Funktion `generate()`, die auf Client-Seite zwei Matrizen `A` und `B` als 2D double Arrays (oder vergleichbares) generiert. Die Matrizen haben folgende Größe: `A` hat die Größe `m` x `l` und `B` die Dimensionen `l` x `n`, mit `m+1=l=n+1`, so dass wir *fast* quadratische Matrizen erzeugen und nur einen Größenparameter `l` verwenden.
  Der Funktion `generate` sollen folgende Argumente übergeben werden
  a. l (int) : definiert die Größe der Matrizen
  b. sparsity (double) in [0,1] : spezifiziert den (durchschnittlichen) Anteil der Zellen der Matrix mit dem Attributwert 0 (diesmal als Zahl)

$$\begin{pmatrix} 3 & 2 & 1 \\ 1 & 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 0 & 1 \\ 4 & 0 \end{pmatrix} = \begin{pmatrix} 7 & 8 \\ 9 & 1\cdot 2 + 0\cdot 1 + 2\cdot 0 \end{pmatrix} = \begin{pmatrix} 7 & 8 \\ 9 & 2 \end{pmatrix}$$

A B C

Abbildung 1 Beispiel einer Matrixmultiplikation aus der Wikipedia (Link)

- **Import der Matrizen in das DBMS (0.5 Punkte)**: Erstellen Sie zwei Tabellen mit dem Schema `A(i,j,val)` und `B(i,j,val)` und importieren Sie ihre per `generate()` erschaffenen Matrizen. Hierbei verweist a_{ij} auf eine Zelle in `A` (also `A[i][j]`). Das Tupel `(i,j,val)` wird nur dann gespeichert, wenn `val` $\neq$ `0` gilt.

A

i	j	val
1	1	3
1	2	2
1	3	1
2	1	1
2	3	2

i	j	val
1	1	1
1	2	2
2	2	1
3	1	4

B

Abbildung 2 Visualisierung der Matrizen nach aus Abb.1 nach Import in die Datenbank.

- **Wahl des Toy Beispiels für Korrektheitstests (0.5 Punkte):** Wählen Sie ein hinreichend komplexes Beispiel. Berechnen Sie von Hand die Daten der importierten Tabelle und das Ergebnis der Matrixmultiplikation C. Die Matrizen müssen den Wert 0 enthalten.
- **Implementierung von Ansatz 0 (0.5 Punkte):** Implementieren Sie einen Ansatz, der eine Matrixmultiplikation auf Client Seite berechnet. Dieser Ansatz dient dazu den Overhead zu quantifizieren, den das DBMS (trotz) Optimierungen erzeugt. Sie dürfen hierfür vorgefertigte Bibliotheken verwenden. Achten Sie in diesem Fall aber darauf, dass der Algorithmus tatsächlich über alle Spalten und Zeilen iteriert, um die Vergleichbarkeit sicherzustellen. D.h. vor allem keine Algorithmen mit sub-kubischer Laufzeit, wie Strassen.
- **Implementierung von Ansatz 1 (0.5 Punkte):** Implementieren Sie den Ansatz aus „Example 2.1 Matrix Multiply with Sparse Representation" des beigefügten Lehrbuchs. Das Ergebnis `C` soll hierbei **nicht** persistent als Tabelle gespeichert werden. Sie können den Ansatz von Client Seite oder direkt im DBMS ausführen. Beachten Sie, dass im Benchmark in Phase 2, analog zum ersten Projekt, viele solche Ausführungen nötig sind.
  Der Ansatz besteht im Wesentlichen aus der Ausführung der folgendes SQL-Anfrage:

```
SELECT A.i, B.j, SUM(A.val*B.val)
FROM A, B
WHERE A.j = B.i
GROUP BY A.i, B.j;
```

Phase 2 – Abschluss des Projekts (2.5 P)

- **Alternativer Import für Ansatz 2 (0.5 Punkte):** Dies ist ein vorbereitender Schritt für die Implementierung von „Example 2.2 Matrix Multiply with Vector Representation". Der Unterschied zu der Repräsentation der Matrizen aus Phase 1 ist, dass nun ein Tupel einer gesamten Zeile bzw Spalte entspricht. Zuvor entsprach ein Tupel einer Zelle der Matrix. Konkret, wird eine Matrix *zeilenweise* gespeichert, die andere *spaltenweise*: `A(i,row)` und `B(B,col)`.
  Für die Wahl des Datentyps von `row` bzw. `col` bieten sich folgende 3 Wege an. Wählen Sie einen hiervon:
  1. Empfohlen: Implementierung als Array. Viele Datenbanksystem unterstützen bereits Arrays[1], die einfach verwendet werden können.
  2. Nicht empfohlen: Erzeugen von großen Tupeln $(i, val_1, \ldots, val_m)$, so dass jedes Attribut einer Zelle entspricht. Dies entspricht der Modellierung in erster Normalform.
  3. Schwierigste Lösung: Implementieren Sie ein User Defined Typ (UDP)[2] `vector`, den Sie dann Mittels Funktionen wie im Lehrbuch beschrieben verwenden. Ob Sie dabei auf externe Sprachen (Python, …) oder PostgreSQL's eigenes pgsql zurückgreifen bleibt Ihnen überlassen. Dies ist mit Abstand die schwierigste Lösung, weshalb es hierfür Bonuspunkte gibt, wie unten beschrieben.
- **Ansatz 2 (0.5 Punkte):** Implementieren Sie einen Ansatz, der wie „Example 2.2 Matrix Multiply with Vector Representation" das Ergebnis der Matrixmultiplikation berechnet. Für die Berechnung einer Zelle in `C` bietet es sich an, eine UDF `dotproduct()` zu verwenden. Beachten Sie, dass das Ergebnis einer Matrixmultiplikation aus Ansatz 1 und 2 identisch sein muss. D.h. insbesondere, dass `C` nicht wieder in die Vektordarstellung überführt werden soll.
- **Benchmark Definition (0.5 P):** Definieren einen Benchmark, der die drei Ansätze für verschieden Matrixgrößen L und sparsity Werte S vergleicht. Definieren Sie sinnvoll gewählte

[1] https://www.postgresql.org/docs/current/arrays.html
[2] https://www.postgresql.org/docs/current/xtypes.html

Werte, so dass alle Kombinationen |L| * |S| für Sie testbar sind. Halten Sie dabei folgende Hinweise ein:

   a. Als Matrixgröße `l` bieten sich Zweierpotenzen ab 2^3 an. Das natürliche Maximum ist, dass die Daten noch in den Hauptspeicher ihres Systems passen.
   b. Sparsity hat in diesem Projekt weniger Gewicht als im ersten Projekt, daher soll die Matrix im Allgemeinen dichter besetzt sein, z.B. `s` in [0.1,0.2,..,0.9].
   c. Sie dürfen frei entscheiden, ob Sie als Metrik Matrixmultiplikation pro fester Zeiteinheit oder die Zeit für x Matrixmultiplikationen verwenden. Idealerweise, sollte die Messzeit pro Kombination `(l,s)` zwischen einer und 5 Minuten betragen.

- **Auswertung (0.5 P):** Führen Sie Ihren Benchmark aus, und stellen Sie das Ergebnis grafisch dar. Was ist ihr Fazit?
- Geben Sie zusätzlich den Code, sowie die Vorstellungspräsentation über Blackboard ab.

Bonuspunkte für UDPs

Für die Verwendung von UDPs in Phase 2 erhalten Sie Bonuspunkte wie folgt:

- Verwendbarkeit des Datentyps `vector` (1 P): Sie erhalten 1 Bonuspunkt, sofern der Datentyp korrekt verwendbar ist. Dies ist der Fall, wenn die `input` und `output` Funktionen definiert sind. Der Inhalt einer Zeile / Spalte also korrekt angezeigt werden kann. Orientieren Sie sich hierfür an der PostgreSQL Doku unter (Link).
- Nutzung externen Sprache (1 P): Definieren Sie eine Funktion `dot_product()` in einer externen Sprach wie folgt:

```
CREATE FUNCTION dot_product(vector, vector)
   RETURNS double
   AS 'filename'
   LANGUAGE C IMMUTABLE STRICT; --Use a language of your choice
```

  Die Funktion soll eine Zelle der Ergebnismatrix `C` berechnen.

Für die Realisierung dieser Lösung wird empfohlen zuerst das Beispiel `complex.sql` and `complex.c` im `src/tutorial` Verzeichnis nachzuvollziehen.

Bewertungsmatrix

Bis wann	Was	#Punkte	Σ Phase
Ende Phase 1	Datengenerator für Matrizen mit Sparsity	0.5	
	Import der Matrizen in das DBMS z.B. als `A(i,j,val)`	0.5	
	Wahl des Toy-Beispiels	0.5	
	Implementierung von Ansatz 1 (Korrektheit wird am Toy Bsp. gezeigt)	0.5	
	Implementierung der Matrixmultiplikation in der Programmiersprache ihrer Wahl (Ansatz 0)	0.5	2.5
Ende Phase 2	Import in das DBMS als UDT oder Array z.B. als `A(i,row) und B(B,col)`	0.5	
	Implementierung von Ansatz 2 (Korrektheit wird am Toy Bsp. gezeigt)	1	
	Definition des Benchmarks	0.5	
	Ausführung und Auswertung des Benchmarks	0.5	2.5
Bonus	Erstellen eines UDT für columns / rows der Matrix	1	
	Implementierung der `dotproduct()`−Funktion in einer externen Sprache (z.B. C oder Python)	1	

Sie erhalten Sie oben angegebenen Punkte, wenn zum Ende der Phase vollständig erfüllt sind, sonst erhalten Sie Anteilig Punkte. Kleinere Nacharbeiten und Korrekturen nach Phase 1 werden individuell vereinbart, so dass die Gesamt-Punkte dennoch erreicht werden können. Nacharbeiten nach Phase 2 sind nicht zulässig.