Performance Analyse der Optimierung von Datenbankabfragen in der HANA Calculation Engine

Projektarbeit 2

im Rahmen der Prüfung zum

Bachelor of Science (B.Sc.)

des Studienganges Wirtschaftsinformatik

an der Dualen Hochschule Baden-Württemberg Mannheim

von

Jared Heinrich

Abgabedatum:	26. August 2024
Bearbeitungszeitraum:	06.05.2024 - 25.08.2024
Matrikelnummer, Kurs:	5101479, WWI22SEA
Ausbildungsfirma:	SAP SE Dietmar-Hopp-Allee 16 69190 Walldorf, Deutschland
Unternehmensbetreuer:	Rainer Agelek
Wissenschaftlicher Betreuer:	Prof. Dr. Hans-Henning Pagnia

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema:

> *Performance Analyse der Optimierung von Datenbankabfragen in der HANA Calculation Engine*

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Mannheim, den 24. Juni 2024

Heinrich, Jared

Abstract

- Deutsch -

Das ist der Abstract.

Inhaltsverzeichnis

Abkürzungsverzeichnis

CE Calculation Engine

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

1. Einleitung

2. Grundlagen

2.1. Calculation Engine

2.2. Ausführung von Datenbankabfragen

Datenbankabfragen werden in der Calculation Engine (CE) in drei Schritten ausgeführt. Zuerst wird die Calculation View instanziiert. Anschließend wird dieses mithilfe verschiedener Methoden optimiert. Dieses optimierte Modell wird dann auf der Datenbank ausgeführt.

Wo genau werden die Abfragen ausgeführt? Was gehört alles zur CalcEngine?

2.3. Verschiedene Arten von Performance Analyse

Benchmarking

Profiling

2.4. Notwendige Grundlagen aus der Graphentheorie

Definition 1 gibt eine Definition für einen gerichteten Graphen nach (Vgl. Knebl 2021, S.220).

> **Definition 1.** *Ein gerichteter Graph ist ein Tupel* $G = (V, E)$. V *heißt Menge der Knoten.* E *heißt Menge der gerichteten Kanten. Es gilt* $V \neq \varnothing$ *und* $E \subset V \times V \setminus \{(v, v) | v \in V\}$. *Zwischen zwei Knoten* $u, v \in V$ *gibt es eine Kante mit dem Anfangspunkt* u *und dem Endpunkt* v, *wenn* $(u, v) \in E$.

Definition 2 gibt eine Definition für einen Pfad in einem Graphen nach (Vgl. Knebl 2021, S.221f).

Definition 2. *Ein Pfad P in G ist eine Folge von Knoten $v_0, \ldots, v_n$. Dabei muss $\forall i \in \{0, \ldots, n-1\} : (v_i, v_{i+1}) \in E$ gelten. v_0 heißt Anfangspunkt von P, v_n heißt Endpunkt von P und n heißt Länge von P.*

Definition 3 gibt nach (Vgl. Knebl 2021, S.222) eine Definition für Zyklen in einem gerichteten Graphen und definiert den Begriff des azyklischen Graphen.

Definition 3. *Ein Pfad heißt geschlossen, wenn $v_0 = v_n$. Ein geschlossener Pfad heißt einfach, wenn $\forall i, j \in \{0, \ldots, n-1\}, i \neq j : v_i \neq v_j$ gilt. In einem gerichteten Graphen heißt ein einfach geschlossener Pfad mit $n \geq 2$ auch Zyklus. Ein Graph heißt azyklisch, wenn er keine Zyklen besitzt.*

Definition 4 definiert den Begriff des gewichteten Graphen nach (Vgl. Knebl 2021, S.253).

Definition 4. *G heißt gewichtet, wenn es eine Abbildung $g : E \to \mathbb{R}$ gibt, welche jeder Kante ein Gewicht zuordnet. Für $e \in E$ heißt $g(e)$ Gewicht von e.*

Definition 5 definiert die Begriffe Kindknoten, Elternknoten und Subknoten.

Definition 5. *Für zwei Knoten c und p gilt, c ist Kindknoten von p, wenn $(p, c) \in V$. Umgekehrt gilt, wenn $(p, c) \in V$, p ist Elternknoten von c. c ist Subknoten von p, wenn $\exists P : (v_0 = p) \wedge (v_n = c)$.*

2.5. Statistische Grundlagen

In Definition 6 wird das metrische Merkmal nach (Vgl. Stocker und Steinke 2017, S.24) definiert.

Definition 6. *Bei einem metrisch skaliertem Merkmale werden einzelne Ausprägungen anhand einer Zahlenskala bewertet, häufig in Verbindung mit einer Maßeinheit. Sie*

> *lassen sich anhand der Größe ordnen und vergleichen. Außerdem kann man die Abstände zwischen verschiedenen Ausprägungen messen und interpretieren.*

Definition 7 definiert die, im Kontext dieser Arbeit gleichbedeutenden, Begriffe Durchschnitt, Mittelwert und Arithmetisches Mittel nach (Vgl. Stocker und Steinke 2017, S.52f).

> **Definition 7.** *Das arithmetische Mittel von n metrisch skalierten Beobachtungswerten* $x_1, \ldots, x_n$*, genannt* $\bar{x}$*.*
>
> *Es gilt:* $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$

Definition 8 beschreibt den Begriff der Varianz nach (Vgl. Fisher 1919, S.399).

> **Definition 8.** *Die empirische Varianz von n metrisch skalierten Beobachtungswerten* $x_1, \ldots, x_n$*, genannt* $\tilde{s}^2$*, ist der Durchschnitt der quadratischen Abweichung von* x_i *von* $\bar{x}$*.*
>
> *Es gilt:* $\tilde{s}^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2$

3. Aktueller Stand der Performance Analyse in der Calculation Engine

In der CE werden bereits verschiedene Wege genutzt, um die Performance und das Verhalten der HANA Datenbank zu analysieren.

3.1. Google Benchmark

Google Benchmark ist ein Open Source Benchmarking Tool von Google, welches es einem ermöglicht einzelne Funktionen in C++ zu benchmarken. Dazu kann man ähnlich zu den meisten Test Frameworks drei verschiedene Codeabschnitte definieren.

Der Hauptabschnitt definiert was genau im Benchmark untersucht werden soll. Diese legt die für die Messung relevante Logik fest.

Der Setup Abschnitt wird einmal vor jeder Ausführung des Benchmarks aufgerufen. In dieser werden die Voraussetzungen für die Ausführung des Hauptabschnitts geschaffen. Man könnte sie beispielsweise nutzen, um Testdaten für den Benchmark zu generieren oder zu laden, da er nicht bei den Messungen beachtet wird.

Der Teardown Abschnitt wird nach jeder Ausführung des Benchmarks aufgerufen. Dieser beeinflusst, wie der Setup Abschnitt, die Messung nicht.

Des Weiteren bietet Google Benchmark die Option, einen Benchmark mehrmals mit unterschiedlichen Parametern durchzuführen. Um beispielsweise die Auswirkung der Größe des Testdatensatzes auf das Ergebnis zu beobachten.

Abbildung 3.1 zeigt eine Beispielhafte Google Benchmark Ausgabe. Benchmark ist dabei der Name des Benchmarks und hinter dem Schrägstrich die Größe des Inputs. Time ist die durchschnittliche Dauer einer Ausführung des Hauptabschnitts über alle Iterationen hinweg. CPU ist die durchschnittliche CPU Zeit einer Ausführung des Hauptabschnitts über alle Iterationen hinweg. Iterations ist die Anzahl der durchgeführten Wiederholungen

des Hauptabschnitts. Legt man keine Anzahl fest, wird die Anzahl der Wiederholungen anhand der durchschnittlichen Dauer einer Iteration und der Varianz der Dauer über alle Iterationen hinweg. (Vgl. Google 2024)

```
-------------------------------------------------------------
Benchmark                      Time           CPU Iterations
-------------------------------------------------------------
BM_RemoveDetachedNodes/2       4 us         4 us       189823
BM_RemoveDetachedNodes/8       9 us         9 us        78165
BM_RemoveDetachedNodes/64     58 us        58 us        12106
BM_RemoveDetachedNodes/512   469 us       468 us         1515
BM_RemoveDetachedNodes/4096 4237 us      4234 us          166
BM_RemoveDetachedNodes/8192 10026 us     10019 us           60
```

Abbildung 3.1.: Google Benchmark Ausgabe

Google Benchmark wird in der CE meistens genutzt, um das Verhalten der Laufzeit bestimmter Funktionen in künstlich generierten Testfällen zu vergleichen. Hierbei wird jedoch keine HANA Instanz benötigt, da keine Operationen auf einer Datenbank ausgeführt werden. Es werden folglich auch keine Testdatensätze für die Datenbank benötigt, sondern nur Daten, auf welchen man die zu analysierende Methode aufrufen kann. Solche Performance Tests sind also im Vergleich zu Tests, welche auf eine HANA Instanz mit Testdatensätzen zugreifen, relativ einfach. Da sie sich weniger Voraussetzungen haben, und nur auf einen kleinen Teil der Logik fokussiert sind.

3.2. HANA Profiler

Eine weitere Methode die Performance und das Verhalten von Software zu analysieren ist, das bereits in Abschnitt 2.3 beschriebene, Profiling. In der CE wird ein eigener Profiler verwendet. Auf diesen kann entweder manuell oder im Code zugegriffen werden. Dabei sind die wichtigsten Befehle `profiler clear` um die aktuellen Profilinginformationen zurückzusetzen, `profiler start` um den Profiler zu starten, `profiler stop` um den Profiler zu stoppen und `profiler print` um die gesammelten Profilinginformationen auszugeben.

Die Ausgabe erzeugt dabei ist dabei zwei gewichtete gerichtete azyklische Graphen, nach Abschnitt 2.4, von denen einer die Verteilung der CPU-Zeit und der andere die Verteilung der Wartezeit, der aufgerufenen Funktionen beinhaltet. Im Folgenden werden die beiden Graphen CPU-Graph und Warte-Graph genannt. Abbildung 3.2 stellt eine mögliche Ausgabe des HANA Profilers dar. Die folgende Beschreibung gilt für den CPU- als auch den Warte-Graphen. Jeder Knoten des Graphen spiegelt eine zur Laufzeit des Profilers aufgerufene Funktion wider. Jeder Knoten beinhaltet drei Informationen. Den Namen der aufgerufenen Funktion, sowie den Wert I und den Wert E. I ist der Anteil der Gesamtzeit, welcher von diesem Knoten und all seinen Subknoten benötigt wurde. E ist der Anteil der Gesamtzeit, welcher von diesem Knoten benötigt wurde. Folglich gilt für alle Knoten $I \geq E$. Die Kindknoten eines Knoten K sind die Funktionen, welche von K aufgerufen wurden.

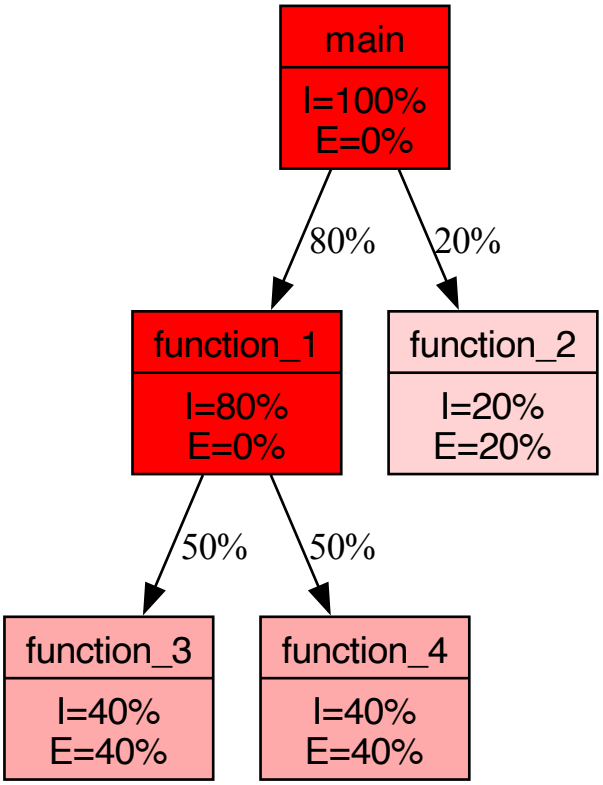

Abbildung 3.2.: Beispielausgabe des HANA Profilers

4. Benchmarking-Konzept

4.1. Messung

Um Informationen über die benötigte CPU-Zeit in verschiedenen Schritten der Optimierung herauszufinden, eignet sich der in Abschnitt 3.2 angesprochene Booss-Profiler sehr gut.

4.2. Testdaten

5. Umsetzung

5.1. Implementation der neuen Methode

6. Verifikation

7. Fazit und Ausblick

A. Anhang

Hier ist der Anhang

Literaturverzeichnis

Knebl, H. (2021). *Algorithmen und Datenstrukturen: Grundlagen und probabilistische Methoden für den Entwurf und die Analyse / Helmut Knebl.* Springer eBook Collection. Springer Vieweg.

Stocker, T. C./ I. Steinke (2017). *Grundlagen und Methodik.* Berlin, Boston: De Gruyter Oldenbourg.

Fisher, R. A. (1919). „XV.—The Correlation between Relatives on the Supposition of Mendelian Inheritance.“ In: *Transactions of the Royal Society of Edinburgh* 52.2, S. 399–433.

Google (2024). *benchmark A microbenchmark support library.* URL: https://google.github.io/benchmark/ (Einsichtnahme: 10. 06. 2024).