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 22. Juli 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. Modelle und Datenbankabfragen

Um Daten einer HANA Datenbank zu analysieren, können sogenannte Kalkulationssichten genutzt werden. Da nur diese Kalkulationssichten in der Arbeit weiter betrachtet werden, werden sie im Folgenden auch als Datenbankabfragen bezeichnet. (Vgl. SAP 2024)

In der Calculation Engine (CE) werden diese Datenbankabfragen durch Modelle dargestellt. Für jede Abfrage wird zuerst das dazugehörige Modell instanziiert. Anschließend wird dieses Modell optimiert, um die Ausführungsdauer der Abfrage zu minimieren. Diese optimierte Abfrage, wird dann auf der Datenbank ausgeführt. Allgemein kann man ein Modell als azyklischen gerichteten Graphen nach Definition 1 und Definition 3 beschreiben. Des Weiteren ist zu beachten, dass dieser Graph nur eine Quelle nach Definition 5 hat, welche auch als Abfrageknoten bezeichnet wird. Diese Definitionen reichen jedoch nicht aus, da zwischen verschiedenen Arten von Knoten unterschieden wird, welche jeweils verschiedene Informationen beinhalten. Allgemein werden die Knoten in der CE in zwei Gruppen unterschieden, Datenquellen und Sichtknoten. Es gibt verschiedene Datenquellen, zur Vereinfachung werden in dieser jedoch Arbeit nur Table-Knoten betrachtet. Deshalb werden Datenquellen im Folgenden auch Tabellenknoten genannt. Die andere Gruppe sind die Sichtknoten. Zu diesen gehören z. B. Projection, Aggregation, Join und Union. Auch hier gibt es zwar noch Weitere, diese Arbeit beschränkt sich jedoch auf diese. Diese Modelle könnte man zwar wie in Definition 1 beschreiben als eine Menge von Knoten und Kanten darstellen, gespeichert wird jedoch eine Liste an Knoten, von welchen jedem seine Kind- und Elternknoten zugeordnet sind. Die Kindknoten werden dabei als Eingangsknoten und die Elternknoten als Ausgangsknoten bezeichnet. Jeder Knoten kann beliebig viele Ausgangsknoten haben, wobei es wie bereits beschreiben nur einen Knoten mit 0 Ausgangsknoten gibt. Die Anzahl der Eingangsknoten unterscheidet sich dabei je nach Knotentyp. Projection- und Aggregation-Knoten haben genau

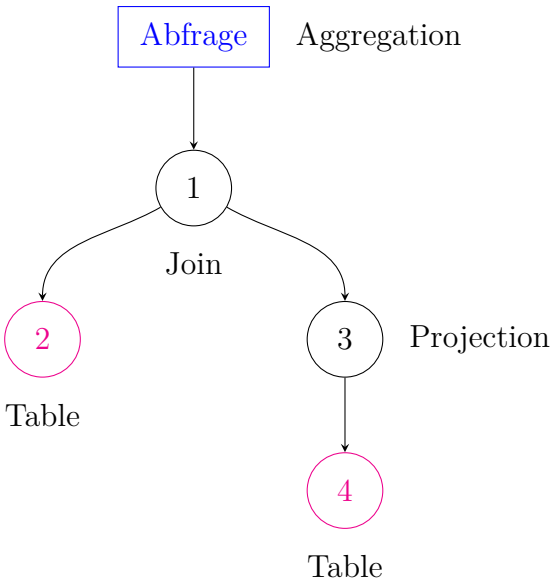


Abbildung 2.1.: Beispiel Modell

einen, Join- und Union-Knoten haben zwei oder mehr und ein Table-Knoten hat keinen Eingangsknoten. Bei den Eingangsknoten kann es sich um Tabellenknoten sowie auch Sichtknoten handeln. (Vgl. SAP 2024, /Create Calculation Views/Supported View Nodes for Modeling Calculation Views)

Was genau ist Aggregation

Abbildung 2.1 zeigt ein einfaches Beispielmodell, welches zur Veranschaulichung dient. Das Modell besteht aus fünf Knoten, dem Abfrageknoten, zwei Sichtknoten und zwei Tabellenknoten. In dem Knoten steht der Name des Knotens und neben ihm steht der Knotentyp. Bis auf den Abfrageknoten haben in diesem Beispiel alle Knoten eine Zahl als Namen. Der Abfrageknoten sowie alle Tabellenknoten sind zusätzlich farblich markiert. Knoten 1 ist ein Join-Knoten. Er hat zwei Eingangsknoten, den Table-Knoten 2 und den Projection-Knoten 3. Der einzige Ausgangsknoten ist in diesem Fall der Abfrageknoten, dieser ist hier vom Typ Aggregation. Der Abfrageknoten könnte jedoch alternativ auch vom Typ Projection sein.

Join nur 2

Unsicher??

Zusätzlich zu den Eingangs- und Ausgangsknoten werden in jedem Knoten noch weitere Informationen gespeichert. Jeder Knoten beinhaltet eine Liste an Sichtattributen, diese legt fest, welche Sichtattribute dieser Knoten an seine Ausgangsknoten weitergibt. Bei einem Tabellenknoten sind die Sichtattribute gleichbedeutend mit den Spalten der Tabelle. Die Sichtattribute des Abfrageknotens sind die Attribute, welche im Ergebnis der Abfrage

enthalten sind. Damit Sichtattribute umbenannt werden können, wird jedem Eingangsknoten eine Liste an Mappings zugeordnet. Hat der Knoten N einen Eingangsknoten E mit einem Mapping, dann bildet dieses ein Sichtattribut von E auf ein Sichtattribut von N ab.

Die verschiedenen Sichtknotentypen sind für verschiedene Operationen zuständig. Manche dieser Knoten haben noch zusätzliche Attribute, um die genaue Art und Weise der Operation festzulegen. Ein Join-Knoten kann genutzt werden, um die Ergebnisse der beiden Eingangsknoten zu verbinden.

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

> **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 (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 Quelle und Senke (vgl. Knebl 2021, S. 306).

> **Definition 5.** *Ein Knoten $q \in V$ heißt Quelle, wenn $\forall p \in V : (p, q) \notin E$, q also nach Definition 6 keine Elternknoten hat. $s \in V$ heißt Senke, wenn $\forall p \in V : (s, p) \notin E$, s also nach Definition 6 keine Kindknoten hat.*

Definition 6 definiert die Begriffe Kindknoten, Elternknoten und Subknoten.

> **Definition 6.** *Für zwei Knoten c und p gilt, c ist Kindknoten von p, wenn $(p, c) \in E$. Umgekehrt gilt, wenn $(p, c) \in E$, p ist Elternknoten von c. c ist Subknoten von p, wenn $\exists P : (v_0 = p) \wedge (v_n = c)$, es also einen Pfad von p nach c gibt.*

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 festgelegt. (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 keine HANA Instanz benötigt, da keine Operationen auf einer Datenbank ausgeführt werden, sondern nur einzelne Funktionen aufgerufen werden. Es werden folglich auch keine Testdatensätze für die Datenbank benötigt, sondern nur Daten, auf welchen man die zu analysierende Funktion aufrufen kann.

Im Vergleich zu anderen Methoden der Performance Messung, die Operationen auf einer HANA-Instanz ausführen, ist ein Google Benchmark relativ einfach. Dies liegt daran, dass sie weniger Voraussetzungen erfordern und sich lediglich auf einen kleinen Teil der Logik konzentrieren.

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 hauptsächlich der „BOOSS-Profiler“, ein in HANA integrierter 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.

Der Profiler kann in diesem Kontext auf zwei verschiedene Arten genutzt werden. Entweder in dem zur Laufzeit des Profilers Anfragen an eine bestehende HANA-Instanz gestellt werden. Oder der Profiler wird innerhalb eines Tests oder Benchmarks aufgerufen und es werden die dort aufgerufen Funktionen gemessen.

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.

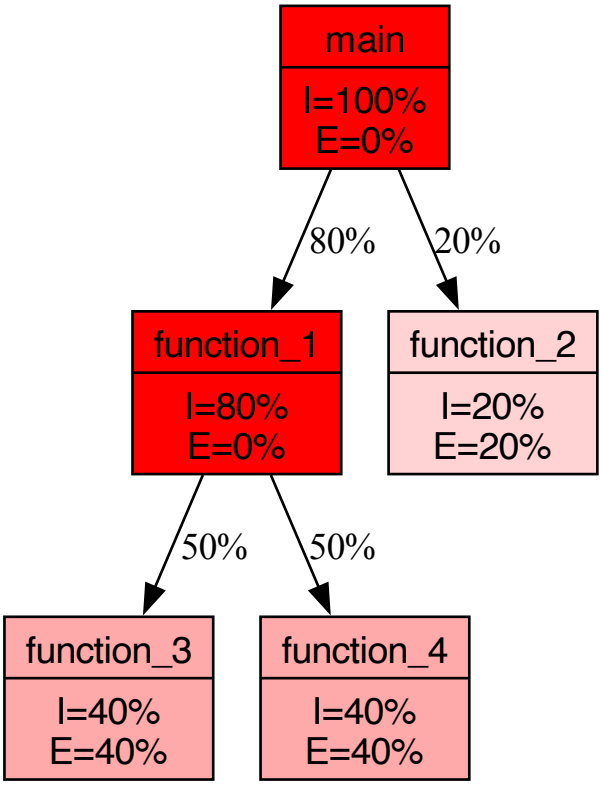


Abbildung 3.2.: Beispielausgabe des HANA Profilers

4. Konzept

Für die Untersuchung des Optimierungsalgorithmus wird ein experimenteller Ansatz statt einem analytischem gewählt, da die experimentelle Vorgehensweise es zum einen einfacher macht sehr komplexe Algorithmen zu untersuchen, zum anderen, eine experimentelle Untersuchung realitätsnähere Ergebnisse liefern kann (vgl. Bartz-Beielstein 2010, S. 3). Hierzu werden in dieser Arbeit zwei unabhängige Variablen betrachtet. Zum einen die Größe und zum anderen der Aufbau des zu optimierenden Modells, diese wurden gewählt, da sie direkt kontrolliert werden können und erwartet wird, dass sie einen großen Einfluss auf die Laufzeit haben (vgl. McGeoch 2002, S. 506). Der Aufbau wird in dieser Arbeit als Art des Modells bezeichnet. Die gemessene abhängige Variable ist die Laufzeit des Optimierungsvorgangs. Unabhängige Variablen sind Variablen, welche aktiv verändert werden, während abhängige Variablen gemessen werden (vgl. Brosius, Haas und Unkel 2022, S. 236). Damit Messergebnisse sinnvoll vergleichen werden können, darf zwischen zwei Messungen nur eine unabhängige Variable verändern werden. (vgl. Brosius, Haas und Unkel 2022, S. 236). Deshalb werden mehrere Messreihen durchgeführt, wobei innerhalb einer Messreihe der Art konstant ist, die Größe jedoch variabel ist. Für jede Art wird eine neue Messreihe begonnen. Störvariablen, also Einflussfaktoren, welche ebenfalls die abhängigen Variablen beeinflussen, jedoch während der Messung unkontrolliert auftreten, ist z. B. die Prozessorauslastung des Rechners, auch welchem die Messung durchgeführt wird (vgl. Brosius, Haas und Unkel 2022, S. 237). Ist der Prozessor weniger ausgelastet, dann ist die gemessene Zeit vermutlich geringer, als wenn der Prozessor stark ausgelastet ist.

Um den Einfluss der Störvariablen möglichst gering zu halten, wird jede Messung n mal wiederholt. Aus diesen Messwerten wird nun ein Konfidenzintervall gebildet, welches mit der Wahrscheinlichkeit $1 - \alpha$ den tatsächlichen Erwartungswert μ enthält. Dazu werden die Messwerte als eine T-verteilte Zufallsvariable X betrachtet, da n aufgrund der Dauer einer Messung nicht sehr groß gewählt werden kann. Die Varianz σ^2 von X ist dabei unbekannt und muss anhand der Stichprobe geschätzt werden, für diese Schätzung gilt: $\hat{\sigma}^2 = S^2$ (vgl Stocker und Steinke 2017, S. 528). Für das $(1 - \alpha)$-Konfidenzintervall ergibt sich deshalb nach (vgl. Stocker und Steinke 2017, S. 533):

$$[\bar{X} - t_{n-1,1-\alpha/2}\sqrt{S^2/n}, \bar{X} + t_{n-1,1-\alpha/2}\sqrt{S^2/n}]$$

Dabei ist $\bar{X}$ der Mittelwert der Stichprobe und S^2 die korrigierte Stichprobenvarianz (vgl. Stocker und Steinke 2017, S. 59, 502).

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{X})^2$$

Für die verschiedenen Arten von Modellen, werden Modelle, von reellen Abfragen betrachtet, um aus diesen, allgemeine Regeln festzulegen, mit welchen man Modelle variabler Größe aber derselben Art erzeugen kann. Die Modelle werden für die Messung künstlich erzeugt, um die unabhängigen Variablen gezielt verändern zu können. Auf die reellen Abfragen wird dabei zurückgegriffen, um die Relevanz der Messung, für reelle Szenarien zu gewährleisten. (Vgl. McGeoch 2002, 500f)

Diese Messdaten werden genutzt, um festzustellen, wie sich die Laufzeit der Optimierung für Modelle bestimmter Art bei steigender Größe verhalten. Dabei ist besonders interessant, ob sich die Laufzeit zur Größe linear verhält, beziehungsweise ob es stärker oder schwächer ansteigt. Für die Optimierung des Algorithmus sind nun die Modellarten interessant, bei welchen die Laufzeit stärker als linear ansteigt, da diese Modellarten ein besonders hohes Potenzial haben lange Laufzeiten zu verursachen. Um genauer herauszufinden, in welchem Teil des Optimierungsalgorithmus besonders viel Zeit benötigt wurde, werden diese Modelle nochmals mithilfe des in Abschnitt 2.3 beschriebenen Profilings untersucht. Dazu werden mehrere Modelle dieser Art optimiert, während der Profiler protokolliert, in welchen Methoden sich wie lange aufgehalten wurde. Anschließend muss beurteilt werden, ob die Zeit, welche in dieser Methode benötigt wird, erwartbar ist oder sie geringer sein sollte. Beziehungsweise, ob es eine Möglichkeit gibt diesen Teil des Algorithmus zu beschleunigen.

5. Umsetzung

5.1. Benchmarking

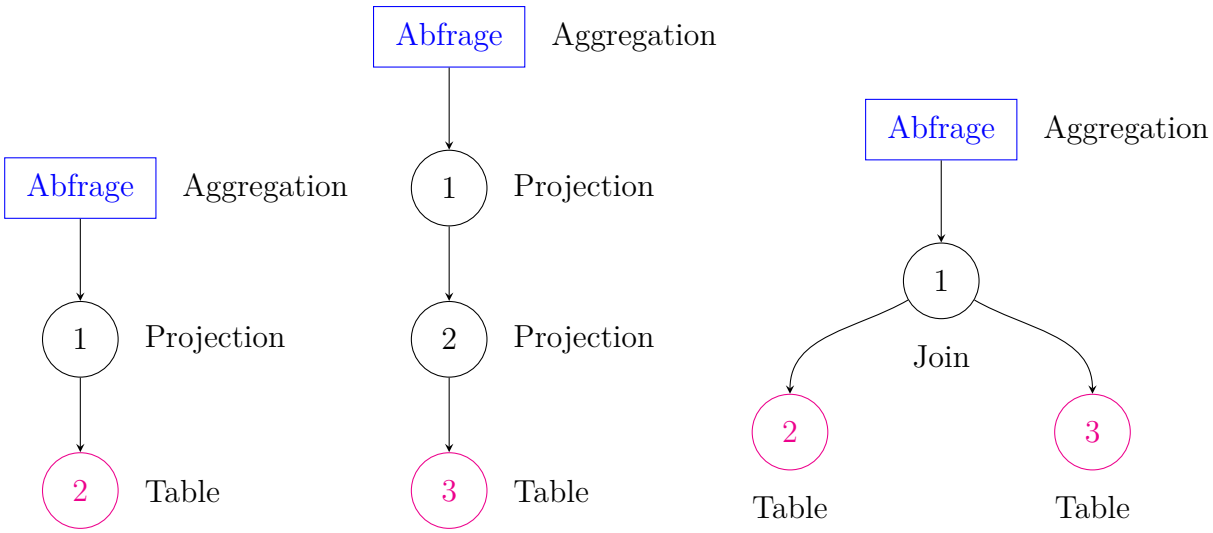
Da sich die Arbeit auf die Optimierungsfunktion der CE beschränkt, kann sich auch bei den Messungen auf diese beschränkt werden. Um die Leistung einer bestimmten Funktion zu untersuchen, eignet sich, das in Abschnitt 2.3 beschriebenen, Benchmarking. Somit sind die Messungen präziser auf diese Funktion ausgerichtet und es wird Aufwand gespart, welcher auftreten würde, wenn man die Untersuchungen anhand einer HANA-Installation durchführen würde. Genauer wird das in der CE verwendete Benchmarking-Framework Google Benchmarks genutzt. Dieses bietet die Möglichkeit eine Messung mit mehreren Parametern und mehreren Variationen von diesen durchzuführen. Es kann z. B. für zwei Parameter Größe und Art jeweils eine Menge, G und A, an Werten festgelegt werden, für welche eine Messung ausgeführt werden soll.

$$G = \{2; 4; 8\} \qquad A = \{j; p\}$$

$$K = G \times A = \{(2, j); (4, j); (8, j); (2, p); (4, p); (8, p)\}$$

Das kartesische Produkt K ist eine Menge von Tupeln (g, a). Jedes Tupel spiegelt eine Kombination von Parametern wider, für welche eine Messung durchgeführt wird (vgl. Ebbinghaus 2021, S. 50). Für alle $(g, a) \in K$ wird das Modell der Art a und der Größe g optimiert und die Dauer dieser Optimierung gemessen. Dieses Modell wird im Folgendem als Modell (g, a) bezeichnet.

Die Parameter sind folgendermaßen definiert. Für jede Art wird eine Funktion definiert, welche ein Modell dieser Art zurückgibt. Die Größe ist ein Wert n, welcher an diese Funktion übergeben wird. Was genau eine Größe von n bedeutet ist dabei von Art zu Art unterschiedlich. Es kann also sein, dass das Modell $(4, j)$ mit Größe 4 und Art j mehr Knoten hat als das Modell $(4, p)$, obwohl sie dieselbe Größe haben. Das spielt jedoch



(a) Beispiel 1 (b) Beispiel 2 (c) Beispiel 3

Abbildung 5.1.: Beispiel Modelle

keine Rolle, da diese Messungen lediglich dazu dienen, zu untersuchen, ob die Laufzeit linear oder stärker anwächst. Es ist jedoch wichtig, dass die Anzahl der Knoten, mit der Größe linear anwächst, da sonst die Kurven nicht mehr vergleichbar währen. Zwei Modelle sind also von gleicher Art, wenn sie mit der gleichen Vorgehensweise erzeugt wurden. Abbildung 5.1 zeigt verschiedene Modelle. Von diesen könnten z. B. Abbildung 5.1a und Abbildung 5.1b von der gleichen Art sein, da diese erzeugt wurden, indem ein Aggregation-Knoten, n Projection-Knoten und ein Table-Knoten hintereinander gehängt werden. Dabei könnte n die Größe sein und $n + 2$ die Anzahl der Knoten.

```
1 BENCHMARK(BM_Optimizer)
2   ->Apply(BenchOptimizerArguments)
3   ->Repetitions(20)
4   ->Unit(benchmark::kMicrosecond);
5
```

Code-Ausschnitt 5.1.: My Listing

Code-Ausschnitt 5.1

Beschreibu der unter-suchten Arten (Co-deaus-schnitte anpassen und

5.2. Profiling

6. Verifikation

7. Fazit und Ausblick

A. Anhang

Hier ist der Anhang

Literaturverzeichnis

Bartz-Beielstein, T., Hrsg. (2010). *Experimental methods for the analysis of optimization algorithms*. Berlin [Heidelberg]: Springer. ISBN: 978-3-642-02537-2.

Brosius, H.-B./ A. Haas/ J. Unkel (2022). *Methoden der empirischen Kommunikationsforschung: eine Einführung*. 8., vollständig überarbeitete und erweiterte Auflage. Wiesbaden [Heidelberg]: Springer VS. ISBN: 978-3-658-34195-4 978-3-658-34194-7.

Ebbinghaus, H.-D. (2021). *Einführung in die Mengenlehre*. 5. Auflage. Berlin [Heidelberg]: Springer Spektrum. ISBN: 9783662638651.

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

Knebl, H. (2021). *Algorithmen und Datenstrukturen: Grundlagen und probabilistische Methoden für den Entwurf und die Analyse*. 2., aktualisierte Auflage. Wiesbaden [Heidelberg]: Springer Vieweg. ISBN: 978-3-658-32714-9 978-3-658-32713-2.

McGeoch, C. C. (2002). „Experimental Analysis of Algorithms“. In: *Handbook of Global Optimization: Volume 2*. Hrsg. von Pardalos, P. M./ Romeijn, H. E. Boston, MA: Springer US, S. 489–513. ISBN: 978-1-4757-5362-2. DOI: 10.1007/978-1-4757-5362-2_14.

SAP (2024). *SAP HANA Cloud, SAP HANA Database Modeling Guide for SAP Web IDE Full-Stack*. URL: https://help.sap.com/docs/hana-cloud-database/sap-hana-cloud-sap-hana-database-modeling-guide-for-sap-web-ide-full-stack/creating-graphical-calculation-view (Einsichtnahme: 09. 07. 2024).

Stocker, T. C./ I. Steinke (2017). *Statistik: Grundlagen und Methodik*. Berlin, Boston: De Gruyter Oldenbourg. ISBN: 978-3-110-35388-4.