ulm university universität ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Organisation und Management von Informationssystemen

Benchmark-basierte Vergleichsbarkeitsanalyse von OLAP Datenbanken

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Nikolai Schuster
nikolai.schuster@uni-ulm.de

Gutachter:
Prof. Dr.-Ing. Dr. h.c. Stefan Wesner

Betreuer:
Dr. Jörg Domaschka
Benjamin Söll

2021

Inhaltsverzeichnis

1 Einleitung

Heutzutage sammelt fast jedes Unternehmen große Mengen von Daten. Die strukturierten Daten werden unter Anderem in Datenbanken gespeichert, die einen wichtigen Teil der heutigen technologischen Infrastruktur von Unternehmen bilden. Um einen Nutzen aus den Daten zu ziehen, bedarf es häufig einer Analyse. Diese Analyse erlaubt es Unternehmen ihre operationale Performanz, Marktentwicklungen, etc. zu untersuchen. Solche Analysen können sehr umfangreich und zeitaufwendig sein, sodass spezielle Datenbanken benötigt werden, die für analytische Aufgaben optimiert sind.

Nun wäre es wünschenswert, wenn sich mittels simpler Internetrecherche schnell Datenbanken für verschiedene Anwendungsszenarien finden ließen. Verschiedene Projekte arbeiten mit unterschiedlich großen Datenmengen, nicht jede Anwendung hat dieselben Performanzansprüche und in der Regel steht kein unlimitiertes Budget für Anschaffungen zur Verfügung. Es hat sich aber gezeigt, dass solche Informationsquellen nicht existieren.

Es gibt viele Datenbanken, deren Hersteller angeben, dass sie für analytische Zwecke besonders gut seien. Sie werben häufig mit selbst erstellten Benchmarks- und Ergebnissen oder geben Performanzmetriken ohne nachvollziehbare Quelle an. Es stellt sich die Frage, wie aus diesem Angebot passende Datenbanken ausgewählt werden können. Wie schon angemerkt, gibt es viele verschiedene Einsatzszenarien mit unterschiedlichen Anforderungen an die Datenbank(en), sodass diese Anforderungen bei der Auswahl berücksichtigt werden müssen. Weiterhin benötigt es einer Evaluierung um zu verifizieren, dass die ausgewählten Datenbanken für die jeweilige Aufgabe auch tatsächlich geeignet sind.

Diese Arbeit beschreibt die Erstellung eines Prozesses zur Auswahl von Datenbanken mit anschließender Performanzbewertung. Dabei werden verschiedene Anforderungen an die Datenbanken berücksichtigt und eine geeignete Methode zur Evaluierung untersucht. Um die Validität des Prozesses festzustellen, werden die einzelnen Schritte des Prozesses

konkret für die Profect Deutschland GmbH durchgeführt, um den Prozess direkt mittels eines echten Anwendungsfalls zu testen. Dazu werden passende Datenbanken ausgewählt, mit der ermittelten Evaluierungsmethode auf die Probe gestellt und die Ergebnisse bewertet. Um die Verständlichkeit und Durchführbarkeit zu ermitteln und Verbesserungspotential zu erkennen, wird der Prozess von anderen Personen durchgeführt und deren Erkenntnisse ausgewertet.

2 Einführung

Diese Arbeit wird beim Unternehmen Profect Deutschland GmbH durchgeführt. Die Profect bietet eine Software an, die es Nutzern ermöglicht Daten zu analysieren. Dazu können Abfragen mit Hilfe einer grafischen Oberfläche erstellt werden, die vom Programm in SQL-Abfragen übersetzt werden. Da diese Abfragen sehr umfangreich werden können und die zu untersuchenden Daten stetig wachsen, muss die im Hintergrund arbeitende Datenbank entsprechenden Anforderungen genügen. Sie muss die Abfragen schnell verarbeiten können, um flüssiges Arbeiten zu ermöglichen. Dabei muss selbstverständlich der Preis berücksichtigt werden. Die aktuell verwendete Datenbank, Exasol, genügt zwar allen Anforderungen, kostet aber auch relativ viel. Bis zu 200 GB Rohdaten können mit der Community Edition kostenlos verwaltet werden, darüber bis zu 1 TB kosten schon 1.999€ / Monat. Dagegen sind z.B. Open Source Datenbanken kostenfrei. Deshalb besteht die Frage, ob es andere Datenbanken gibt, die eine zumindest vergleichbare Leistung zu einem geringeren Preis liefern.

Die Suche im Internet bringt zwar einige mögliche Kandidaten hervor, allerdings verspricht jeder Anbieter viel, ohne dass das direkt verifiziert werden kann:

> **Apache Ignite**: „Distributed Database For High-Performance Applications With In-Memory Speed“ - [14]

> **Exasol**: „Nutzen Sie die unübertroffene Performance unserer Analysedatenbank, wo immer Sie möchten – in der Cloud, lokal auf Ihren eigenen Servern oder in einer hybriden Umgebung. Noch nie war es einfacher und kostengünstiger, Unternehmenserkenntnisse in kürzester Zeit in echten Mehrwert zu verwandeln.“ - [12]

Eine echte Vergleichbarkeit kann durch Benchmarks ermöglicht werden [4]. Die einzige zentrale und unabhängige Plattform für Datenbank-Benchmarks, die im Zuge der Recherche gefunden wurde, ist vom Transaction Processing Performance Council (TPC) (http://tpc.org/). Sie haben eigene Benchmarks erstellt und veröffentlichen unabhängig verifizierte Ergebnisse (s. Kapitel 5).

Auch haben manche Anbieter oder Dritte selbst Benchmarks mit den Datenbanken durchgeführt und die Ergebnisse veröffentlicht [47, 32]. Aber es gibt nicht viele Ergebnisse, die mit demselben Benchmark erzielt wurden. So sind auf der TPC-Homepage z.B. nur für vier Datenbank (verschiedene Versionen von Exasol und Microsoft SQL Server jeweils zusammengezählt) Ergebnisse zum TPC-H Benchmark zu finden [55].

Außerdem wurden diese Benchmarks in der Regel mit unrealistischer Hardware und Datenmengen durchgeführt. Viele Unternehmen bzw. einzelne Projekte arbeiten nicht mit Datenmengen größer als 100 GB. Die in den Benchmarks eingesetzte Hardware wäre dann nicht wirtschaftlich für diese Aufgaben [51]. Es fehlen Benchmarkergebnisse für realistischere Anwendungsszenarien.

Ein weiteres Problem sind Benchmarks, bei denen nicht alle relevanten Informationen veröffentlicht werden [48, 6]. Ohne die Spezifikationen der benutzen Hardware und/oder des benutzen Benchmarks, sind die Ergebnisse nicht vergleichbar. Somit kann auch keine Aussage getroffen werden, wie die getestete Datenbank in einem eigenen Anwendungsfall agieren würde. Darüber hinaus werden manchmal nur teilweise Ergebnisse veröffentlicht. Dadurch werden die Stärken der Datenbank hervorgehoben und die Schwächen versteckt [43]. Ein solches Ergebnis wäre potentiell völlig irreführend.

Daraus ergibt sich als Problemstellung: Wie kann bewertet werden, ob eine analytische Datenbank den eigenen Anforderungen genügt? Es wird also eine möglichst objektive Bewertung gebraucht um eine passende Entscheidung treffen zu können. Damit ergibt sich die Forschungsfrage: Wie kann eine OLAP-Datenbank bewertet werden?

2.1 Ziel der Arbeit

Das Ziel der Arbeit ist herauszuarbeiten, wie OLAP-Datenbanken bewertet und verglichen werden können. Daraus soll ein Prozess entstehen, der es ermöglicht weitere oder in der Zukunft erscheinende Datenbanken zu testen. Dabei soll auch die Auswahl der Datenbanken nach bestimmten Kriterien berücksichtigt werden, um gegebenenfalls Datenbanken auszuschließen, bevor sie getestet werden. Dieser Prozess soll flexibel sein, sodass er auf verschiedene Anwendungsszenarien und mit unterschiedlichen Anforderungen anwendbar ist, um jeweils eine geeignete Datenbank bestimmen zu können. Der Prozess soll getestet werden, indem er benutzt wird um verschiedene Datenbanken auszuwählen und anschließend zu testen.

2.2 OLAP

OLAP steht für *Online Analytical Processing* und wurde von Edgar F. Codd 1993 als Begriff eingeführt [44]. Dabei geht es um die Analyse von Daten unter verschiedenen Aspekten, häufig als Teil der Business Intelligence/des Decision Supports. Codd definierte ursprünglich zwölf Regeln, später auf 18 erweitert, die Systeme erfüllen müssten, um als ein OLAP-System zu gelten. Um die Definition eines OLAP-Systems greifbarer zu machen, erstellten Nigel Pendse und Richard Creed die FASMI-Definition [40]. Diese Regeln sollen unabhängig von der/den genutzten Technologie/n sein, d.h. es ist nur relevant, dass das System die Regeln erfüllt, nicht wie.

- **Fast**: Ein System soll schnell Antworten liefern, in der Regel unter fünf Sekunden, maximal 20 Sekunden für aufwändige Anfragen. Längere Zeiten verschlechtern die Nutzbarkeit für Endnutzer deutlich.

- **Analysis**: Es soll einfach möglich sein, alle Arten von Analyse durchführen zu können.

- **Shared**: Es soll sicher, also ohne Konflikte oder Seiteneffekte, möglich sein, mit mehreren Nutzern gleichzeitig auf dem System zu arbeiten.

- **Multidimensional**: Es muss möglich sein, eine multidimensionale Betrachtung der Daten vorzunehmen.

- **Information**: Alle Daten, die zur Analyse notwendig sind, müssen jederzeit bereit gestellt werden können.

Eine OLAP-Datenbank muss also ermöglichen, dass ein System, das diese Datenbank nutzt, diesen Regeln entsprechen kann. Das bedeutet insbesondere für die Datenbank, dass sie auch aufwändige Queries schnell bearbeiten können muss. Sie sollte außerdem in der Lage sein, möglichst alle Arten von Queries bearbeiten zu können, um verschiedenen Analyseanforderungen gerecht zu werden. Sie muss mit mehreren Nutzern gleichzeitig umgehen können, wobei auch in diesem Fall eine ausreichende Performanz gewährleistet werden muss.

3 Anforderungen

Eine Datenbank muss bestimmte Eigenschaften haben, um für ein Projekt oder ein Unternehmen geeignet zu sein. Die benötigten Eigenschaften hängen von den Anforderungen des Projekts/Unternehmens ab. Somit ist es essentiell, dass diese Anforderungen bei der Auswahl einer Datenbank bzw. der Datenbanken berücksichtigt werden.

Anforderungen (an eine Software) unterteilen sich in funktionale und nicht-funktionale. Funktionale Anforderungen definieren was eine Software konkret machen/können muss oder welche Eigenschaften sie besitzen soll. Nicht-funktionale Anforderungen beschreiben allgemeiner wie sich die Software verhalten soll [31]. Beiden Arten von Anforderungen können direkt abhängig vom jeweiligen Anwendungsfall sein und müssen somit für jeden Anwendungsfall neu bestimmt werden. Oder sie sind allgemein wichtig, weil sie grundsätzlich Kosten und Mehraufwand senken.

3.1 Funktionale Anforderungen

Die funktionalen Anforderungen ergeben sich direkt aus dem Anwendungsfall für den die Software, hier eine Datenbank, benutzt wird. Im Gegensatz zu den nicht-funktionalen Anforderungen (s. 3.2) gibt es keinen Erfüllungsgrad der Anforderungen, sondern die Entscheidung ist binär. Z.B. ist eine Datenbank entweder ein Open Source (Definition nach [26]) Projekt oder nicht, es gibt dabei keine Abstufung. Entsprechend eignen sie sich zur Vorauswahl einer Datenbank, weil es in der Regel möglich sein sollte festzustellen, ob die Anforderung erfüllt wird, ohne die Datenbank selbst daraufhin testen zu müssen. Die Anforderungen wurden basierend auf eigenen Erfahrungen in der Projektentwicklung, sowie derer erfahrener Mitarbeiter von Profect, bestimmt. Die folgenden Kriterien stellen

speziell Anforderungen eines Softwareprojekts von Profect dar. Sie werden im Weiteren dazu verwendet entsprechende Datenbanken zum Testen auszuwählen.

Diese Anforderungen sind absolut notwendig. Nicht-Erfüllung bedeutet direkt, dass die Datenbank nicht für das Projekt geeignet ist.

SQL-fähig: Die Datenbank muss mit SQL (der konkrete Dialekt ist nicht entscheidend) bedienbar sein. Profect Entwickler beherrschen alle SQL und SQL wird in der Regel an Universitäten unterrichtet. Wenn die Datenbank mit SQL zu bedienen ist, hat so jeder die Möglichkeit diese zu nutzen ohne vorher Zeit in das Erlernen einer neuen Sprache investieren zu müssen. Oder kann zumindest auf die umfassende Kompetenz anderer Mitarbeiter zurückgreifen. Außerdem ist ein integraler Bestandteil der Software von Profect ein Querybuilder. Eine Datenbank ohne SQL Support liese sich nur mit großem Mehraufwand integrieren, da ein Übersetzer von SQL zu jeweiligen anderen Sprache nötig wäre.

Joins: Insbesondere Joins müssen unterstützt werden. Joins sind essentiell für Profects Software. Um Joins herum zu arbeiten würde zu viel Arbeit von der Datenbank auf die Anwendung umlagern.

Keine Datenwürfel: Datenwürfel (engl. data cubes, auch OLAP-Würfel) sind mehrdimensionale Arrays von Daten. Die Daten können so leicht aus verschiedenen Dimensionen betrachtet werden [15]. Allerdings erfordert die Erstellung von Datenwürfeln vorheriges Wissen über die Art der Abfragen. Da, wie erwähnt, bei Profects Software beliebige Abfragen erstellt werden können, sind Datenwürfel nicht zielführend. Eine Datenbank sollte deshalb nicht ausschließlich auf Datenwürfeln basieren.

Die folgenden Anforderungen sind wünschenswert, da sie den Arbeitsaufwand senken und damit verbundene Kosten oder direkt Kosten senken.

Docker Image: Docker Images erlauben die Datenbank meist auf beliebigen Systemen zu starten ohne auf irgendwelche Abhängigkeiten achten zu müssen. So lässt sich die Datenbank leicht in den Softwarestack integrieren. Die Existenz eines Docker Images ist nicht unbedingt notwendig, da diese auch selbst erstellt werden können. Allerdings erfordert dies Mehraufwand einer qualifizierten Person.

Open Source: Open Source bedeutet, dass der Quellcode frei einsehbar und veränderbar ist [26]. Dies erlaubt es selbstständig Änderungen an der Datenbank vorzunehmen, wenn es Bedarf gibt. Das sorgt für mehr Flexibilität. Außerdem ist Open Source Software in der Regel frei erhältlich.

Dokumentation: Eine umfassende Dokumentation erleichtert den Einstieg und erlaubt es bei Problemen schnell eine Lösung zu finden.

Diese Anforderungen werden bei der Auswahl der Datenbanken nicht berücksichtigt. Sie stellen nur weitere mögliche Anforderungen dar.

Data Ingestion: Es kann bestimmte Anforderungen für die Art, wie Daten eingelesen werden, geben. Evtl. reicht es CSV-Dateien einlesen zu können oder es ist eine Anbindung an Spark notwendig.

SaaS: Software as a Service hat den Vorteil, dass man sich nicht selbst um Hosting und Wartung der Datenbank kümmern muss. Die anfallenden Kosten können durch Einsparungen bei eigenem Arbeitsaufwand und Training für entsprechende Kompetenzen gedeckt werden.

3.2 Nicht-funktionale Anforderungen

Für nicht-funktionale Anforderungen kann nicht im Vorhinein bestimmt werden, ob diese erfüllt werden. Sie beschreiben, was ein System leisten muss und es muss gemessen werden, inwieweit diese Anforderungen erfüllt werden. Unterschiedliche Anforderungen erfordern unter Umständen andere Tests [45], sodass klar sein muss, was relevant ist und getestet werden muss.

Performanz: Performance ist eine der wichtigsten Eigenschaften. Sie kann noch weiter unterteilt werden in
– Query-pro-Stunde / Durchsatz: Die Anzahl an Queries, die in einer bestimmten Zeiteinheit durchgeführt werden können
– Ausführungszeit / Antwortlatenz: Die Zeit, die für die Ausführung einer einzelnen Query benötigt wird.

Beide Ausprägungen hängen auch von einander ab. Eine niedrigere Antwortlatenz erhöht direkt den Durchsatz. Allerdings kann der Durchsatz erhöht werden, in dem mehrere Queries parallel bearbeitet werden. Dadurch stehen jeder einzelnen Abfrage in der Regel weniger Ressourcen zur Verfügung und die Antwortlatenz steigt. Ebenso können einer einzelnen Query überdurchschnittlich viele Ressourcen zugewiesen werden, sodass die Antwortlatenz sinkt, aber für andere Abfragen weniger Ressourcen existieren, sodass der Durchsatz sinkt. Profects Software hat meist nur wenige gleichzeitige Nutzer, weshalb Durchsatz nicht entscheidend ist. Für Profect ist Antwortlatenz der wichtigste Aspekt. Schnelle Antworten ohne Wartezeit sind angenehmer in der Nutzung und ermöglichen ein flüssiges Arbeiten. Performanz soll im weiteren Verlauf der Arbeit getestet werden.

Die folgenden Anforderungen werden in dieser Arbeit nicht getestet und sind nur der Vollständigkeit halber aufgeführt.

Wartbarkeit/Nutzbarkeit: Wartbarkeit/Nutzbarkeit sagt, wie viel Aufwand es benötigt, eine Anwendung aufzusetzen und betriebsbereit zu halten. Sie wird von vielen Faktoren beeinflusst. Einige wurden in 3.1 schon explizit erwähnt, Docker Image, Dokumentation, Saas. Hier können aber noch viele andere Aspekte einspielen, die den allgemeinen Umgang erleichtern und Zeit einsparen, z.B. Existenz von Continuous Integration/Delivery Unterstützung/Tools, selbstständiges Anlegen von Indizes und Query Optimizer. Dadurch lässt sich Geld einsparen, weil zum Einen weniger Expertise/Training nötig sind, wodurch Zeit und Kosten für die Ausbildung gespart werden. Zum Anderen ist weniger direktes Zeitinvestment für Wartungsaufgaben nötig.

Skalierbarkeit: Skalierbarkeit ist die Eigenschaft sich erhöhten Anforderungen anzupassen. Wenn die Ausführung von Queries zu lange dauert, müssen zusätzliche Hardware-Ressourcen zur Verfügung gestellt werden. Dafür gibt es zwei Möglichkeiten [1]:
– Horizontale Skalierung (scale out): Beschreibt das Hinzufügen von Servern. Ein zusätzlicher Server kann ohne Downtime für die Datenbank in Betrieb genommen werden. Außerdem sorgt dies dafür, dass im Falle eines Ausfalls eines Servers die anderen weiterhin zur Verfügung stehen und den Service aufrecht erhalten. Allerdings erfordern mehrere Server mehr Wartungsaufwand und zusätzliche Software zum Managen (Load Balancer).
– Vertikale Skalierung (scale up): Ist das Hinzufügen von zusätzlicher Hardware zu einem existierenden Server. Das ist billiger als einen kompletten Server hinzuzufügen und ist einfacher zu managen als mehrere Server. Wenn nur ein einzelner Server existiert, ist downtime

erforderlich für die Erweiterung. Zusätzlich ist ein solches Upgraden nicht unbeschränkt möglich.

Elastizität: Elastizität ist die Möglichkeit bei kurzfristigen Nutzungsspitzen, zum Teil automatisch, zusätzliche Ressourcen zuzuschalten und danach wieder zu entfernen. Dies ermöglicht, dass die Datenbank immer voll operativ ist [36]. So können alle Nutzer die Software uneingeschränkt weiter nutzen. Es müssen aber nicht permanent weitere Ressourcen mit laufenden Kosten bereit gestellt werden.

Verfügbarkeit: Verfügbarkeit beschreibt inwieweit ein System jederzeit erreichbar und operativ ist [23]. Dies ist wichtig für eine Datenbank, damit Nutzer bzw. die benutzte Software an die benötigten Daten kommt. Für kritische Software ist eine höhere Verfügbarkeit notwendig, sodass entsprechende Vorkehrungen getroffen werden müssen, um Ausfällen vorzubeugen und unabhängig weiterlaufende Datenbankeninstanzen bereit zu stellen.

4 Datenbankauswahl

Für die Benchmarks wurden Datenbanken zum Testen benötigt. Dazu wurden in Kapitel 3 konkrete Anforderungen bestimmt, die eine potentielle Datenbank erfüllen soll, um für diese Arbeit in Frage zu kommen. Vor dieser Arbeit bestand die Annahme, dass die Suche nach Datenbanken, die bestimmte Anforderungen erfüllen, völlig problemlos sein sollte. Es stellte sich allerdings heraus, dass dies nicht ohne Weiteres möglich war. So gibt es keine Plattform, wo z.B. eine Suche nach dem Schlagwort „OLAP Database “oder ähnlichem möglich ist. So musste die Suche verschiedene Quellen benutzen, um Kandidaten zu finden. Dabei war es nur eingeschränkt möglich, direkt nach den funktionalen Anforderungen zu filtern. Stattdessen musste für die potentiellen Kandidaten die Erfüllung der funktionalen Anforderungen teilweise auf den Herstellerwebseiten oder in deren Dokumentation recherchiert werden.

4.1 Relevante Technologien

Um die Anzahl der zu betrachtenden Datenbanken einzuschränken und weil insbesondere die Database of Databases (s. 4.2) nur technische Filter ermöglicht, wurden Technologien recherchiert, die sich für analytische Datenbanken bewährt haben. Dabei haben sich Spaltenorientierte Datenbanken [50] und In-Memory-Datenbanken [50, 33, 41] besonders hervorgetan. Diese Technologien sind nicht notwendig für eine analytische Datenbank. Aber Datenbanken, die wenigstens eine dieser Technologien besitzen, werden, wie sich herausstellte, als analytische Datenbanken angegeben. Deshalb können diese Begriffe als Suchfilter benutzt werden.

Spaltenorientierte Datenbank: Lange Zeit wurden die Daten einer Datenbank zeilenweise gespeichert. So war es möglich einen einzelnen Eintrag mit einer Schreiboperation auf die Festplatte zu schreiben. Für transaktionenorientierte Datenbanken ist eine solche Schreibop-

timierung von Vorteil, weil ständig neue Datensätze persistiert werden müssen. Beim Lesen existiert allerdings der Nachteil, dass immer eine vollständige Zeile ausgelesen werden muss, auch wenn nur einzelne Felder (Spalten) von Interesse sind. Für analytische Abfragen sind vor allem Leseoperationen relevant, sodass diese zeilenweise Speicherung eher nachteilig ist. Dafür bietet sich spaltenweise Speicherung an. Für Leseoperationen müssen nur die Spalten betrachtet werden, die tatsächlich benötigt werden. Ein weiterer Vorteil ist, dass die Werte innerhalb einer Spalte viel homogener sind als solche in verschiedenen Spalten. Die Werte innerhalb einer Spalte sind vom selben Datentyp, z.B. Integer, Char, und die Spalten enthalten teilweise nur wenige verschiedene Werte. Dadurch lassen sich die Daten viel besser komprimieren, wodurch Speicherplatz gespart wird [50].

In-Memory-Datenbank: In den letzten Jahren sind die Kosten von RAM stark gefallen, sodass es realistisch ist, genügend RAM zu haben um auch sehr große Datenmengen im RAM zu halten. Dadurch ist es nicht mehr nötig, ständig kleine Segmente von der Festplatte in den RAM zu laden und sich die zeifaufwändigen I/O-Zugriffe zum Laden von Daten auf der Festplatte in den RAM zu sparen. Dadurch kann die Performanz deutlich verbessert werden [33, 41].

4.2 Kandidatensuche

Es gibt sehr viele Datenbanken, DB-Engines [27] zählt über 370 auf [28]. Alle von Hand zu überprüfen, ob sie sich laut eigener Aussage als analytische Datenbank eignen, wäre sehr viel Arbeit. Auf DB-Engines werden Datenbanken anhand ihrer Popularität eingestuft, gemessen über die Häufigkeit der Suchergebnisse in Google und Bing. Datenbanken können anderen gegenübergestellt und kriterienweise verglichen werden. Abb. **??** [29] zeigt eine Gegenüberstellung von Exasol und Apache Ignite und einige der Kriterien anhand derer die Datenbanken verglichen werden können. Es ist nicht möglich nach bestimmten Schlagworten zu suchen. Auch gibt es nicht für alle Datenbanken vollständige Einträge, so existieren z.B. nur jeweils eine Seite mit minimalen Informationen für PrestoDB und Trino. Deshalb eignet sich die Webseite eher zum Vergleichen bekannter Datenbanken als zum Suchen neuer Datenbanken für einen bestimmten Zweck. Außerdem können Datenbankeinträge schnell auf bestimmte Eigenschaften überprüft werden, wie Lizenz, SQL-Unterstützung, Replikationsmechanismen, etc.

Vergleich der Systemeigenschaften EXASOL vs. Ignite

Redaktionelle Informationen bereitgestellt von DB-Engines		
Name	**EXASOL** X	**Ignite** X
Kurzbeschreibung	High-performance, in-memory, MPP database specifically designed for in-memory analytics.	Apache Ignite is a memory-centric distributed database, caching, and processing platform for transactional, analytical, and streaming workloads, delivering in-memory speeds at petabyte scale.
Primäres Datenbankmodell	Relational DBMS	Key-Value Store Relational DBMS
Datenschema	ja	ja
Typisierung	ja	ja
XML Unterstützung	nein	ja
Sekundärindizes	ja	ja
SQL	ja	ANSI-99 for query and DML statements, subset of DDL

Abbildung 4.1: Vergleich Exasol vs. Apache Ignite

Die einzige andere Plattform zum Suchen nach Datenbanken, die währender der Recherche für diese Arbeit gefunden wurde, ist die Database of Databases [17]. In der Database of Databases können Suchergebnisse nach bestimmten Kriterien gefiltert werden, die aber auf technische Aspekte beschränkt sind. So ist es möglich, nach der Art des Datenmodells oder der Kompressionsart zu suchen, aber nicht allgemein nach analytischen Datenbanken. Auch sind hier für manche Datenbanken nur minimale Informationen vorhanden, z.B. Exasol, sodass diese in der Suche nicht auftauchen, weil das entsprechende Kriterium nicht getagged ist (Joins für Exasol). Kürzlich wurden zum Exasoleintrag die Tags „Decomposition Storage Model (Columnar) “und „In-Memory “hinzugefügt, sodass Exasol darüber suchbar ist (Stand 5.2.2022).

In der Database of Databases wurde nach Datenbanken gesucht mit den folgenden Kriterien, jeweils angegeben in der Form „Kategorie - Ausprägung “(s. Abb. **??**:

- Kombination Project Types - Open Source
- Joins - (Hash Join oder Nested Loop Join oder Sort-Merge Join)
- Storage Architecture - In-Memory
- Storage Model - Decomposition Storage Model (Columnar)

Die drei Ausprägungen für Joins sind die drei häufigsten Algorithmen für Joins. Storage Architecture und Storage Model beziehen sich jeweils auf die Technologien aus 4.1. Es

wurden jeweils Kombinationen aus den Suchbegriffen gebildet, wobei Open Source und eine der Join-Optionen jeweils festgehalten wurden, also war eine Kombination z.B. Open Source, Hash Join, Columnar. Mit diesen Suchen wurden diese Datenbanken gefunden:

In-Memory Columnar	In-Memory		Columnar		
bel. Join	Hash	Nested	Hash	Nested	Sort-Merge
BlazingSQL	AlaSQL	ActorDB	AresDB	C-Store	Apache Druid
Clickhouse	Compass	ArangoDB	DuckDB	Greenplum	
Apache Hive		CovenantSQL	Apache Kylin	DeepDB	
MariaDB CS		ts-sql	LucidDB	Spark SQL	
NoisePage			MonetDB	Apache Drill	
Hyrise			QuestDB	InfiniDB	
PrestoDB			Solr	WiredTiger	
			Trafodion		

Tabelle 4.1: Database of Databases Suchergebnisse

Dabei sind in der Tabelle 4.1 in den Spalten nur jeweils die Datenbanken angegeben, die mit jeder weiteren Suchkombinationen hinzugekommen sind.

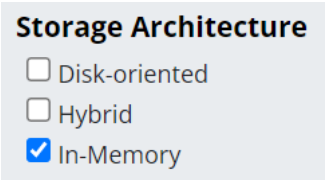


Abbildung 4.2: Suchkriterium in der Database of Databases

Ein paar weitere Datenbanken haben sich noch aus anderen Quellen ergeben. Diese sind nicht in der Database of Databases Suche gefunden worden, obwohl jeweils ein Eintrag für sie existiert. Eine Mitarbeiterempfehlung war Apache Ignite. Auf Wikipedia gibt es eine Liste von open-source spaltenorientierten Datenbanken, wobei CrateDB gefunden wurde [57]. Als Nebeneffekt der Literaturrecherche haben sich noch ergeben: TiDB [21], SnappyData [19].

4.3 Kandidatenauswahl

Zunächst musste die Anzahl der Datenbanken deutlich reduziert werden. Viele der Datenbanken ließen sich aufgrund verschiedener Faktoren schnell aussortieren (s. Tabelle 4.2).

Datenbank	Grund der Aussortierung	Datenbank	Grund der Aussortierung
BlazingSQL	GPU basiert	Spark SQL	hält nicht selbst Daten
AresDB	GPU basiert	PrestoDB	hält nicht selbst Daten
Apache Hive	Hadoop basiert	NoisePage	für selbstfahrende Autos
Trafodion	Hadoop basiert	Hyrise	für Forschung/Ausbildung
Compass	Entwicklungsende 2009	AlaSQL	für Webbrowser/JavaScript
LucidDB	Entwicklungsende 2009	CovenantSQL	Blockchain basiert
C-Store	Entwicklungsende 2008	ts-sql	Hobby Projekt
DeepDB	Entwicklungsende 2017	DuckDB	embedded Datenbank
InfiniDB	Entwicklungsende 2014	QuestDB	für Time-Series Daten
ArangoDB	kein SQL	Solr	Search Engine
Apache Druid	kein SQL	WiredTiger	NoSQL Datastore
Apache Kylin	Datenwürfel basiert		

Tabelle 4.2: Grund der Aussortierung

Die GPU basierten Datenbanken wurden ausgeschlossen, weil dafür extra Hardware hätte angeschafft werden müssen. Hadoop basiert und „nicht selbst Daten haltend “hätten extra Systeme erfordert, um die Daten zu verwalten. Datenbanken, die nicht mehr entwickelt werden, wurden ausgeschlossen, weil es nicht sinnvoll erschien potentiell veraltete Software zu testen. Datenbanken, die keine SQL-Unterstützung haben oder auf Datenwürfeln basieren, wurden schon über die Anforderungen ausgeschlossen, konnten aber nicht schon vorher herausgefiltert werden. WiredTiger ist nur eine Storage Engine. Die restlichen Datenbanken haben scheinbar sehr spezielle Anwendungsgebiete und passten deshalb nicht in den Umfang dieser Arbeit.

Alle übrigen Datenbanken erfüllten die gestellten Anforderungen (s. Abschnitt 3.1). Im Prinzip waren alle geeignete Testkandidaten. Aber um den Aufwand für diese Arbeit überschaubar zu halten, wurden fünf exemplarisch ausgewählt. Diese sind in der Abbildung grün hinterlegt, die abgelehnten rot. In der Spalte *Kommentare* ist der Ablehnungsgrund aufgeführt.

- TiDB und SnappyData fielen raus, weil sie keine CLI besitzen, die im weiteren zum Interagieren mit der Datenbank benutzt werden soll.

- ClickHouse besitzt keinen Queryoptimizer, hier müssten Queries evtl. händisch optimiert werden.
- Greenplum scheint auf den ersten Blick in die Dokumentation ein kompliziertes Setup aus mehreren Nodes zu erfordern.
- ActorDB ist eine völlig unbekannte Größe und belegt im DB-Engines Ranking den geteilten letzten Platz.

Als Testkandidaten wurden entsprechend die folgenden fünf Datenbanken ausgewählt: Apache Ignite, CrateDB, MariaDB (Columnstore), MonetDB und Apache Drill. Hinzu kam außerdem Exasol als Referenzkandidat. Wie in Kapitel 2 erwähnt, ist Exasol bereits eingesetzt worden und hat gute Ergebnisse geliefert. Deshalb eignet sich die Datenbank um ihre Benchmarkergebnisse mit den neu ausgewählten zu vergleichen. Als finaler Kandidat wurde PostgreSQL bestimmt. PostgreSQL ist keine analytische Datenbank. Aber es wäre durchaus interessant herauszufinden, ob analytische Datenbanken wirklich besser geeignet sind für die Datenanalyse und wie viel besser sie gegebenenfalls sind. Da schon eigene Erfahrungen im Umgang mit PostgreSQL existieren, fiel die Wahl auf diese konventionelle Datenbank.

Exasol wird momentan schon bei Profect erfolgreich eingesetzt und bietet sich daher als Referenzkandidat an. Außerdem wirbt Exasol mit „Unsere Analytics-Datenbank bietet Ihnen unübertroffene Performance“ und hält mehrere aktuelle (Stand: 08.01.2022) Top-Platzierungen im TPC-H Benchmark, weshalb davon auszugehen ist, dass Exasol eine gute Performanz hat, an der die anderen Datenbanken gemessen werden können. Allerdings benötigt Exasol eine kostenpflichtige Lizenz, sodass ein adequater, kostenfreier Ersatz interessant wäre.
Exasol benutzt folgende Technologien: In-Memory, MPP, Spaltenorientierte Speicherung, Selbstoptimierung (unter anderem selbstständige Verwaltung von Indexen) [13].

CrateDB baut auf einer NoSQL-Datenbank auf, ist aber voll SQL-fähig (PostgreSQL). Ad-hoc-Abfragen und Datenaggregation sowie -filterung sollen schnell bearbeitet werden. Daten werden sowohl zeilenweise als auch spaltenweise gespeichert. Sie hat sich vor allem durch eine automatische Verwaltung von Indexen und eine ausgezeichnete Dokumentation von den anderen potentiellen Kandidaten abgesetzt. Außerdem besitzt CrateDB eine eigene CLI, Crash-CLI, die das Ansprechen über die Command Line ermöglicht [7].

Apache Ignite ist eine In-Memory Datenbank, die auf einem Key-Value-Store basiert, aber trotzdem mit SQL benutzt werden kann. Von den potenziellen Kandidaten ist sie die bekannteste laut DB-Engines (nach Greenplum). Mit dem Tool sqlline [22] kann Ignite über die Command Line angesprochen werden.

MariaDB ColumnStore ist eine spaltenorientierte Datenbank und als Teil von MariaDB oder getrennt verfügbar. Sie benutzt eine *massively parallel distributed data architecture* [34]. Der Columnstore ist mit MariaDBs CLI kompatibel.

MonetDB ist eine spaltenorientierte Datenbank. Der Optimizer sorge für eine parallele Ausführung der Abfragen, um viele CPU-Cores gleichzeitig zu nutzen. MonetDB kann selbstständig Indizes erzeugen. Die Datenbank unterstützt SQL und besitzt eine eigene CLI [2].

Apache Drill ist eine SQL Query Engine für Hadoop, NoSQL und Cloud Storage. Sie kann Daten aus unterschiedlichen Quellen abfragen und joinen, ohne die Daten selbst laden zu müssen. Drill besitzt eine spaltenbasierte Execution Engine. Drill benutzt als CLI sqlline [22]

PostgreSQL ist eine der bekanntesten Datenbanken laut DB-Engines. Sie wurde ausgewählt, weil der Umgang mit ihr bereits bekannt ist und so wenig zusätzliche Arbeit erfordert. PostgreSQL wird stellvertretend für konventionelle Datenbanken getestet, da es von Interesse ist, ob solche Datenbanken nicht schon ausreichend Performanz bieten für analytische Zwecke.

5 Benchmarkauswahl

Zur Messung der Performanz von Datenbanken sind Benchmarks ein probates Mittel [16]. Dazu muss ein passender Benchmark ausgewählt werden. Dabei muss berücksichtigt werden, dass der Benchmark abbildet, was versucht wird zu testen. In diesem Fall ist das die analytische Auswertung von großen Datenmengen. Es gilt herauszufinden, ob es bereits Benchmarks gibt, die dieser Anforderung genügen oder ob eine eigene Implementierung nötig ist.

5.1 Definition Benchmark

Benchmarks werden benutzt um Prozesse oder Produkte zu evaluieren. Dazu müssen Messkriterien, z.B. Dauer, Preis, bestimmt werden, die zur späteren Bewertung dienen. Es muss außerdem ein Testablauf erstellt werden, wie das Produkt/der Prozess getestet wird. Dabei ist wichtig, dass der Testablauf und die Kriterien klar definiert sind, um Nachvollziehbarkeit und Wiederholbarkeit zu gewährleisten. Dieser Benchmark kann nun auf andere ähnliche Prozesse oder Produkte angewendet werden, um sie zu vergleichen. Die Ergebnisse der Benchmarks können benutzt werden, um den besten Prozess/das beste Produkt hinsichtlich der angewendeten Messkriterien für einen bestimmten Anwendungsfall zu ermitteln oder Optimierungspotenzial zu bestimmen [20].

5.2 Warum ein etablierter Benchmark?

Einen eigenen Benchmark zu erstellen hat den großen Vorteil, dass der eigene Anwendungsfall komplett abgedeckt wird und wirklich relevante Ergebnisse erziehlt werden. Allerdings

ist das auch sehr viel Arbeit und birgt viel Fehlerpotenzial [16]. Es würde also eine hoch qualifizierte Fachkraft benötigt werden, die Zeit und damit Geld investiert. Außerdem gibt es keine Vergleichsergebnisse, da niemand sonst diesen speziellen Benchmark durchgeführt hat. Da es etablierte und anerkannte Benchmarks gibt, die genau für OLAP-Probleme erstellt wurden, werden für diese Arbeit deshalb nur diese Benchmarks in Betracht gezogen, insbesondere auch, da der Fokus mehr auf dem Prozess der Evaluierung von Datenbanken als auf den konkreten Ergebnissen liegt.

5.3 Transaction Processing Performance Council

Das Transaction Processing Performance Council (TPC) wurde 1988 gegründet. Es ist eine Non-Profit Organisation die sich auf Anbietern von Datenbanken zusammensetzt. Ihr Ziel ist die Erstellung von Benchmarks und Überwachung der Durchführung dieser Benchmarks. Das TPC hat für unterschiedliche Anwendungsbereiche von Datenbanken spezialisierte Benchmarks entwickelt. Die Durchführung der Benchmarks wird mit klaren Regeln definiert und vor Veröffentlichung durch eine vom durchführenden Unternehmen unabhängige Person verifiziert. Dadurch soll gewährleistet werden, dass die Ergebnisse vergleichbar sind und sich kein Anbieter von Datenbanken unfairerweise besser als seine Konkurrenz darstellen kann [46].

5.4 TPC-H Benchmark

Der TPC-H Benchmark ist ein Benchmark für entscheidungsunterstützende (decision support, DS) Systeme und wurde vom TPC 1999 vorgestellt [42]. Bei seiner Veröffentlichung wurden für DS noch OLTP Datenbanken benutzt, heutzutage werden jedoch hauptsächlich analytische bzw. OLAP Datenbanken verwendet. TPC-H ist der gängigste Benchmark für OLAP Anwendungen [11] und er werden noch immer neue Ergebnisse veröffentlicht [55].

Beschreibung: Der Benchmark bildet die Datenbank eines Güterlieferanten nach. Dazu können mittels eines bereitgestellten Tools, DBGEN, Daten generiert werden, die sich auf acht Tabellen verteilen (Customer, Lineitem, Nation, Orders, Part, Partsupp, Region,

Supplier). Die Tabellen sind in der dritten Normalenform (3NF) aufgebaut. Die Datenmenge kann mittels eines Scale Factors (1, 10, 30, 100, 300, 1.000, 3.000, 10.000, 30.000 und 100.000 GB) eingestellt werden. Die Benutzung dieser Daten wird durch 22 Queries aus dem Bereich DS dargestellt. Diese sind so aufgebaut, dass sie möglichst alle Daten einbeziehen und verschiedene Anforderungen abdecken. Die Queries können mit dem Tool QGEN für verschiedene SQL-Dialekte generiert werden. Der Ablauf sieht folgendermaßen aus. Für den *Load Test* werden die Daten geladen. Dann werden für den *Power Test* die 22 Queries sequentiell ausgeführt. Jeweils vor und nach dem Power Test wird ein Paar von *Refresh Functions* ausgeführt. Refresh Functions sind Inserts oder Deletes in den Tabellen Orders und/oder Lineitems. Abschließend werden nochmals die 22 Queries durchgeführt jeweils in mehreren parallelen Sessions, wobei die Anzahl vom Scale Factor abhängt. Dies wird *Throughput Test* genannt und soll mehrere gleichzeitige User simulieren. Die jeweilige Reihenfolge der Queries ist in der TPC BENCHMARK H Standard Specification (ist im TPC-H Kit enthalten [54]) Appendix A festgehalten. Während des Throughput Tests müssen zu einem beliebigen Zeitpunkt weitere Refresh Functions ausgeführt werden.

Die Performanz Metrik ist die *TPC-H Composite Query-per-Hour Metric* (QphH@Size, die Preis-Performanz Metrik ist die *TPC-H Price/Performance* ($ /kQphH/@Size).

Für den Power Test wird die Berechnungspower bestimmt:

$$\text{TPC-H Power@Size} = \frac{3600 * SF}{\sqrt[24]{\prod_{i=1}^{i=22} QI(i,0) * \prod_{j=1}^{j=2} RI(j,0) *}},$$

mit $QI(i,0)$ die gemessene Zeit in Sekunden von Query Q_i des Power Tests, $RI(j,0)$ die gemessene Zeit in Sekunden von Refresh Function RF_j des Power Tests, und Size die Größe der Datenmenge für den gewählten Scale Factor SF. Für den Throughput Test wird der Durchsatz berechnet:

$$\text{TPC-H Throughput@Size} = \frac{S * 22 * 3600}{T_s} * SF,$$

mit T_s der Gesamtdauer des Throughput Tests, S der Anzahl paralleler Query-Sessions, und Size die Größe der Datenmenge für den gewählten Scale Factor SF. Insgesamt ergibt sich damit für die TPC-H Composite Query-per-hour Metric:

$$\text{QphH@Size} = \sqrt{\text{Power@Size} * \text{Throughput@Size}}.$$

Damit kann die TPC-H Price/Performance Metric gebildet werden:

$$\text{TPC-H Price-per-kQphH@Size} = \frac{1000 * \$}{\text{QphH@Size}},$$

mit $\$$ der Gesamtkosten des genutzt Systems zur Durchführung des Benchmarks, QphH@Size der Composite Query-per-Hour Metric, und Size die Größe der Datenmenge für den gewählten Scale Factor SF.

Es gibt auch Kritik am TPC-H Benchmark. In [38] wird kritisiert, dass nach [30] das 3NF Schema und der Inhalt mancher Tabellen nicht angemessen seien für ein Data Warehouse. Stonebraker bemerkt, dass TPC-H Benchmarks auf Hardware durchgeführt werden, die viel mächtiger ist, als normale Nutzer benutzen würden. Außerdem fehle ein Test für die Datenladeperformanz. Weiterhin bemängelt er, dass der Benchmark zu komplex und zu schwierig durchzuführen sei [49].

5.5 TPC-DS Benchmark

Der TPC-DS Benchmark ist ein neuerer DS Benchmark vom TPC. Im Jahr 2000 wurde mit der Entwicklung eines Nachfolgers für TPC-H [52] begonnen und 2012 veröffentlicht [53].

Beschreibung: Als Datenbasis wird ein Händler von Gütern nachgestellt. Wiederum existiert ein Datengenerator, der Daten für die Scale Factors 100, 300, 1000, 30.00, 10.000, 30.000 und 100.000 GB erstellen kann. Dabei skalieren die Fact Tables linear und die Dimension Tables nur sublinear, damit die Datenbasis nicht degeneriert für große Scale Factors im Gegensatz zu TPC-H. Die Tabellen sind in einem Snowflake Schema angeordnet, weil dieses sich, zusammen mit Star Schema, in der Industrie als Standard hervorgetan hat [37]. Der Workload besteht aus 99 Queries, die sowohl ad-hoc- als auch Reporting-Queries beinhalten. Dazu kommen zwölf Datenpflege-Queries, bestehend aus Inserts, Deletes und Updates. Für den Benchmark werden zunächst die Daten geladen (*Load Test*) und danach die 99 Queries ausgeführt (*Power Test*). Anschließend folgt zweimal jeweils ein Durchlauf der Queries für mehrere Nutzer, abhängig vom Scale Factor, parallel ausgeführt (*Throughput Test*) und Datenpflege (*Integration Test*). Die jeweilige Reihenfolge der Queries ist in der

TPC BENCHMARK DS Standard Specification (ist im TPC-DS Kit enthalten [54]) Appendix D festgehalten [37].

Die Performance Metric für TPC-DS wird folgendermaßen berechnet:

$$\text{QphDS@SF} = \frac{SF * S_q * 99}{\sqrt{T_{PT} * T_{TT} * T_{DI} * T_L}^A},$$

mit S_q die Anzahl der gleichzeitigen Nutzer, $T_{PT} = T_{\text{SingleUser}} * S_q$, $T_{\text{SingleUser}}$ Ausführungszeit des Power Tests, $T_{TT} = T_{\text{MultiUsers}_1} + T_{\text{MultiUsers}_2}$, $T_{\text{MultiUsers}_s}, s \in \{1, 2\}$ Ausführungszeit des s-ten Throughput Tests, $T_{DI} = T_{\text{DataMaintenance}_1} + T_{\text{DataMaintenance}_2}$, $T_{\text{DataMaintenance}_s}, s \in \{1, 2\}$ Ausführungszeit des s-ten Integration Tests, und $T_L = 0,01 * S_q * T_{\text{Load}}$, T_{Load} Ausführungszeit des Load Tests.

Die Price/Performance Metric wird analog zum TPC-H berechnet:

$$\text{QphDS@SF} = \frac{1000 * \$}{\text{QphH@Size}}.$$

Im Jahr 2015 kam die Version 2 heraus, die ein paar Schwachstellen der Version 1 ausbessern und somit für mehr Akzeptanz sorgen sollte. Unter anderem müssen Queries nicht mehr ACID erfüllen, sondern nur noch BASE, um NoSQL Systeme mit einzubeziehen [56].

ACID ist ein Akronym für die folgenden Begriffe [18]:

- Atomicity: Eine Transaktion wird immer als eine Einheit durchgeführt. Bei einem Fehler werden alle bisherigen Aktionen der Transaktion rückgängig gemacht.
- Consistency: Eine Transaktion überführt die Datenbank von einem konsistenten Status wieder in einen konsistenten Status.
- Isolation: Nach Ausführung von mehreren gleichzeitigen Transaktionen befindet sich die Datenbank im selben Zustand, wie wenn die Transaktionen sequentiell ausgeführt worden wären
- Durability: Nach Durchführung einer Transaktion ist diese persistent.

Das Gegenstück zu ACID für NoSQL-Datenbanken ist BASE [5]:

- Basically Available: Leser oder Schreiben ist immer möglich, auch wenn andere Lese- oder Schreibtransaktionen im Gange sind.

- Soft State: Zu einem gegebenen Zeitpunkt ist die Datenbank nicht notwendigerweise in einem konsistenten Zustand

- Eventually Consistent: Die Datenbank wird irgendwann (möglichst zeitnah) in einem konsistenten Zustand sein.

Bei der Datenpflege kommen keine Updates mehr vor. Der Benchmarkablauf wurde angepasst: Load Test, Power Test (Einzelnutzer), zweimal Throughput Test mit jeweils anschließendem Integration Test (Datenpflege). Außerdem wurden die Performanzmetriken vom arithmetischen zum geometrischen Mittel geändert [56].

Kritik: TPC-H wurde schon für zu komplex befunden. TPC-DS hat deutlich mehr Abfragen, sodass die Komplexität noch gestiegen ist [49]. Außerdem sind die Hardwareanforderungen extrem gestiegen, weil der kleinste Datensatz 100 GB beträgt.

5.6 Star Schema Benchmark

Der Star Schema Benchmark (SSB, SSBM) wurde 2009 vorgestellt und basiert auf dem TPC-H Benchmark. SSB versucht Schwachstellen von TPC-H auszubessern. Die erste Änderung ist die Verwendung eines Star Schemas statt der 3NF, weil das mehr Sinn macht für Data Warehouses [30]. Außerdem wurden die Queries überarbeitet. Zum Einen, um dem neuen Schema zu genügen, zum Anderen um Functional und Selectivity Coverage einzubeziehen [39].

Beschreibung: Analog zu den Abläufen der TPC Benchmarks, ist auch SSB aufgebaut. Die Daten werden geladen (*Load Test*), ein einzelner Durchlauf der Queries (*Power Test*), Datenpflege (*Refresh Test*) und paralleler Durchlauf von zwei Querystreams (*Throughput Test*).

Kritik: Der SSB benutzt simplere Queries. Dadurch werden geringere Anforderungen an einen Queryoptimizer gestellt. Je nach nachdem, was getestet werden soll, kann das vor- oder nachteilhaft sein.

5.7 Auswahl des Benchmarks

Die Hardwareanforderungen des TPC-DS Benchmark sind zu hoch. Weder arbeitet Profect mit Daten in dieser Größenordnung, noch ist es realistisch nur für diese Arbeit ein adequates System zu beschaffen. Somit fällt der TPC-DS raus.

Trotz der umfangreichen Kritik wurde der TPC-H Benchmark ausgewählt. Der SSB soll zwar Probleme des TPC-H Benchmarks ausgleichen, dennoch ist TPC-H der bekanntere Benchmark. Somit sollten mehr potentielle Vergleichsergebnisse zur Verfügung stehen. Außerdem ist der höhere Anspruch, den der TPC-H an die Datenbank stellt, interessant. Da bei Profect eine Software im Einsatz ist, bei der sich Nutzer eigene Abfragen erstellen (vgl. Kapitel 2), können diese nicht beliebig optimiert werden. Entsprechend muss die Datenbank in der Lage sein auch mit solchen Queries umzugehen. Die Kritik, dass die Durchführung des TPC-H Benchmarks zu schwierig sei, gilt gleichermaßen für den SSB. Beide sind ähnlich genug aufgebaut, sodass es in dieser Hinsicht keine signifikanten Unterschiede geben sollte.

6 Durchführung

Die Durchführung beschreibt den Ablauf der Benchmarking-Versuche. Es ist wichtig, dass der Ablauf fest definiert ist, um Nachvollziehbarkeit und Wiederholbarkeit der Versuche und der Ergebnisse zu gewährleisten. Eine häufig benutzte Option für diesen Anwendungsfall sind bash-Skripte. Sie lassen sich leicht erstellen und ändern. Sie ermöglichen Zugriff auf hilfreiche Unix-Programme, wie sed und awk, die dabei helfen können gegebenenfalls Daten zu manipulieren. Außerdem können problemlos Commandline-Programme zur Kommunikation mit den unterschiedlichen Datenbanken eingebunden werden. Die erstellten Skripte[1] können einfach auf verschiedenen Systemen ausgeführt werden, sodass es ohne viel Aufwand möglich ist auf verschiedenen Systemen Tests durchzuführen, sofern Linux vorhanden ist.

6.1 Vorgehensweise

Jede Datenbank soll mit ihrer Standardkonfiguration aufgesetzt werden. Sinnvolle Änderungen an der Konfiguration vorzunehmen erfordert Erfahrung mit der Datenbank. Es ist auch nicht klar, wie eine vergleichbare Menge von Optimierungen an unterschiedlichen Datenbanken aussehen würde. Eine andere Option wäre, jede Datenbank möglichst stark zu optimieren. Dies würde zu viel Aufwand erfordern, sodass die Standardeinstellungen als einzige Möglichkeit bleiben. Es ist auch durchaus interessant, ob die Datenbanken eine gute Performanz liefern ohne zusätzliche Arbeit investieren zu müssen, was hinsichtlich der Wartbarkeit positiv wäre.

Mit jeder Datenbank sollen mehrere Setups getestet werden. Dabei werden die zur Verfügung gestellten Hardwareressourcen (CPUs) und der Scale Factor des Benchmarks variiert.

[1] Die Skripte sind im folgenden Git-Repository zu finden: https://git.profect.de/nsc/nsc_ba_public

Ursprünglich war auch die Variation des RAMs angedacht. Allerdings haben die ersten Benchmarkergebnisse dies nicht sehr sinnvoll erscheinen lassen, weil Exasol mit sehr wenig RAM auskommt, während alle anderen viel mehr benötigen und schon nur unter optimalen Bedingungen mit Exasols Performanz konkurrieren können (s. Kapitel 7). Um Einflüsse etwaiger anderer Prozesse, die im Hintergrund laufen, oder sonstiger Störungen auszugleichen, werden mehrere Durchläufe, 10 zum Start, gegebenenfalls mehr, für jedes Hardware-Scale-Factor-Setup gestartet. Als initiale Scale Factors wurden 1 und 10 ausgewählt, weil dass die beiden kleinsten offiziellen Faktoren sind und es sich so am ehesten abschätzen lies, wo die Leistungsgrenzen der Datenbanken liegen. Die Ergebnisse mit diesen Datenmengen haben gezeigt, dass größere Mengen mit der gegebenen Hardware für die meisten getesteten Datenbanken nicht zu bewältigen sind (s. Kapitel 7). Die Anzahl der CPU-Kerne sind die beiden Extreme, die möglich sind, also 1 Kern oder alle. Das Exasol-Docker-Image ist auf 10 GB Datengröße beschränkt. Scale Factor 10 passt gerade nicht mehr in die Datenbank, sodass Scale Factor 9 verwendet wird. Die angedachten Setups sind:

- unbeschränkt, also so viel, wie das Testsystem erlaubt (s. Sektion 6.5), Scale Factor 1
- unbeschränkt, Scale Factor 9
- 1 CPU, Scale Factor 1
- 1 CPU, Scale Factor 9

In jedem Durchlauf wird jeweils ein Einzel(nutzer)lauf (Power Test), also nur eine Query zur Zeit, durchgeführt, gefolgt von einem Multi(nutzer)lauf (Throughput Test), bei dem Queries für zwei NUtzer parallel ausgeführt werden (vgl. 5.4).

Der TPC-H wird in leicht abgeänderter Form durchgeführt. Es wird auf die Ausführung der Refresh Queries verzichtet. Bei dem Profect Projekt, für das die Benchmarks durchgeführt werden (s. Kapitel 2), gibt es keine regelmäßigen Datenschreibvorgänge. Bei nicht offiziell durchgeführten TPC-H Benchmarks ist dieses Vorgehen nicht unüblich [3], [47].

6.2 Aufbau

Die Datenbanken werden jeweils in einem Docker Container gestartet. Für die Benchmarks wurden mehrere Skripte erstellt: Ein Orchestrations-Skript, das die anderen Subskripte in der richtigen Reihenfolge ausführt, ein Skript zum Starten des Orchestrations-Skripts mit verschiedenen Parametern und neun Subskripte zur Durchführung der nötigen Teilschritte (s. 6.4). Dazu kommen ein paar Verzeichnisse, in denen die Query-Templates für die einzelnen Datenbanken, die Daten- und Querygeneratoren und gegebenenfalls CLIs der Datenbanken liegen.

6.3 Datenbankcontainer

Zum Aufsetzen der Datenbanken wird Docker benutzt [9]. Docker ermöglicht Anwendungen abgekapselt in sogenannten Containern zu starten. Diese Container enthalten alles notwendige für die Anwendung. Um den Prozess noch weiter zu vereinfachen, wurden docker-compose.yaml Dateien erstellt, mit denen die Container konfiguriert und über Docker Compose gestartet werden können [10]. Durch Container wird gewährleistet, dass die Datenbanken auf beliebigen Systemen, sofern Docker installiert ist, laufen und immer unter gleichen Bedingungen ausgeführt werden können. Außerdem ist es so möglich, die benötigte Datenbank zu starten und nach Ende des Benchmarks wieder zu stoppen. So können ungenutzte Datenbanken nicht unnötig Hardwareressourcen verbrauchen.

6.4 Skripte

Im Startskript start_benchmark.sh können die einzelnen Subskripte ausgewählt werden, die laufen sollen. Dies ermöglicht einzelne Schritte wegzulassen, wenn sie nicht nötig sind, weil z.B. schon Daten erstellt wurden oder nur ein Multi-User-Durchlauf getestet werden muss. Außerdem wird die Datenbank angegeben, für die der Benchmark durchgeführt werden soll, der Scale Factor für die Datenmenge, die Anzahl der Benutzer im Multi User Durchlauf, die

Anzahl der Einzel- und Mehrnutzerläufe, und die Information der Master Node bei cratedb bzw. Zertifikat für Exasol.

Das Orchestrations-Skript ruft die einzelnen Subskripte in der korrekten Reihenfolge auf, insbesondere führt es die Single- und Multi-User-Läufe wie vorher eingestellt entsprechend oft auf.

Die Subskripte haben die folgenden Aufgaben:

1. rollout_compile.sh:
    - Kompiliert die Dateien des TPC-H-Kits. Damit stehen der Daten- und der Query-generator zur Verfügung.
    - Wählt die entsprechenden Querytemplates für die spätere Generierung der Queries aus.
2. rollout_gen_data.sh:
    - Generiert Daten, dazu wird generate_data.sh für jeden CPU-Kern des Systems gestartet, um die Generierung zu parallelisieren und dadurch zu beschleunigen.
    - Erzeugt in generate_queries.sh mittels der entsprechenden Query-Templates die durchzuführenden Queries. Die Query-Templates sind syntaktische Anpassungen gemäß dem benutzten SQL-Dialekt der jeweiligen Datenbank. Sie sind aber funktionell identisch.
3. rollout_ddl.sh:
    - Löscht ein evtl schon existierendes Schema inkl. der enthaltenden Tabellen und erstellt anschießend ein neues Schema.
    - Führt die Create-Anweisungen für die Tabellen des Benchmarks aus.
4. rollout_load.sh:
    - Lädt die Daten in die Datenbank.

- Erstellt danach gegebenenfalls Constraints, Primary Key und Foreign Key, sowie Indizes wenn nötig. Die Constraints werden erst nach der Datenbefüllung angelegt, um den Prozess zu beschleunigen. Außerdem können die Constraints zu Problemen führen, wenn die Daten nicht in einer bestimmten Reihenfolge geladen werden.

5. rollout_sql.sh:

   - Führt die Einzelnutzer-Queries durch.
   - Misst und loggt die Zeit für jede Query.
   - Loggt die Auslastung der Hardwareressourcen (CPU und RAM) mittels docker stats.

6. rollout_reports.sh: Fasst die Logs der Einzelnutzerergebnisse zusammen.

7. rollout_multiuser.sh:

   - Startet für jeden Benutzer des Mehrnutzerlaufs multi_user_query_run.sh, das wiederum die Queries für den entsprechenden Nutzer in der richtigen Reihenfolge durchführt.
   - Misst die Zeit für jede Query für jeden Nutzer.
   - Misst und loggt die Gesamtzeit des Mehrnutzerlaufs.
   - Loggt die Auslastung der Hardwareressourcen (CPU und RAM) mittels docker stats.

8. rollout_multiuser_reports.sh: Fasst die Logs der Mehrnutzerergebnisse zusammen und sortiert sie.

6.5 Testsystem

Das Testsystem ist eine VM auf einem internen Server. Auf der VM läuft als Betriebssystem Ubuntu 18.04.6 LTS. Der VM stehen 22 logische CPU-Kerne vom Typ E5-2620V3 mit 2.4 GHz und 128 GB RAM zur Verfügung.

7 Auswertung

Im Folgenden werden die Ergebnisse[1] des TPC-H Benchmarks mit den Datenbanken Exasol, Apache Ignite, CrateDB, MariaDB Column Store, und PostgreSQL ausgewertet. Dabei werden etwaige Probleme beim Aufsetzen der Datenbanken und Durchführen des Benchmarks beschrieben. Außerdem werden die Ausführungszeiten der Queries und die Auslastung der CPU-Kerne und des RAMs analysiert. Anschließend soll ein direkter Vergleich der Datenbanken die beste(n), hier schnellste(n), Datenbank(en) aufzeigen. Weiterhin wird ausgewertet, wie andere Personen die Nutzung des entwickelten Prozesses bewerten, um Verbesserungspotential aufzudecken.

7.1 Auswertung Exasol

Exasol ließ sich ohne Probleme mit der Defaultkonfiguration aufsetzen. Das benutzte Docker-Image erfordert allerdings im Privileged Mode ausgeführt zu werden, was dem Container Root-Zugriff auf dem Hostsystem gibt. Sicherheitstechnisch ist das nicht empfehlenswert für eine Produktionsumgebung, hat allerdings keinen Einfluss auf die hier durchgeführten Tests. Sowohl das Laden der Daten als auch Ausführen der Queries war ohne weiteres möglich. Eine Einschränkung ist, dass das Docker Image auf 10 GB Datengröße beschränkt ist. Dadurch passte der Datensatz mit Scale Factor 10 (Größe 10 GB) gerade nicht mehr rein, weshalb Scale Factor 9 benutzt wurde.

Exasol war in der Lage alle verfügbaren Cores zu nutzen, wie die Auslastung bei den unbeschränkten Durchläufen zeigt. Die durchschnittliche Auslastung war allerdings viel geringer. Exasol musste nur kurzfristig auf viele Cores zugreifen, die meiste Zeit hat weniger

[1] Die Rohdaten sind im folgenden Git-Repository zu finden: https://git.profect.de/nsc/nsc_ba_public

als einer gereicht (s. Tabelle 7.1). Merkwürdig ist, dass bei den unbeschränkten Läufen im Mehrnutzerlauf nur etwa die Hälfte der Cores wie im jeweiligen Einzellauf benutzt wurden.

Setup	S Avg.	S Max.	M Avg.	M Max.
C 1, SF1	0,34	1	1	1,25
C 1, SF9	0,56	1,02	0,71	1,14
unbe., SF1	0,15	10,36	0,29	4,9
unbe., SF9	1,28	22,42	1,18	10,6

Tabelle 7.1: Anzahl genutzter Cores

Wie die Tabelle 7.2 zeigt, betrug die höchste RAM Auslastung 1,83 GB (unbeschränkt, SF9, multi), die niedrigste 1,36 GB (1 Core, SF9, single). Hierbei ist interessant, dass es keinen großen Unterschied in der Belegung gab bei einem unterschiedlichen Scale Factor, was bei einer In-Memory-Datenbank zu erwarten wäre.

Setup	S Avg.	S Max.	M Avg.	M Max.
C 1, SF1	1,41	1,471	1,44	1,5
C 1, SF9	1,36	1,58	1,42	1,6
unbe., SF1	1,42	1,46	1,45	1,51
unbe., SF9	1,57	1,82	1,6	1,83

Tabelle 7.2: RAM-Auslastung in GB

In der Tabelle 7.3 sind die Laufzeiten der unterschiedlichen Setups dargestellt. Dabei wird die Zeit des ersten Durchlaufs jeweils getrennt angegeben. Der erste Durchlauf für jedes Setup dauerte deutlich länger als alle Folgenden. Außerdem brauchte in der Regel der erste Multinutzerdurchlauf weniger Zeit als der erste Einzelnutzerdurchlauf. Das kann am Caching von Queries und/oder der automatischen Erstellung von Indizes liegen. Es fällt auf, dass bei erhöhtem Scale Factor die Bearbeitungszeit nicht linear mitwächst. Der größte Anstieg ist bei einem Core im Mehrnutzerdurchlauf mit Faktor 1,91 bzw. 5,52 für den initialen Durchlauf.

Setup	S initial in s	S avg. Rest in s	M initial in s	M avg. Rest in s
C 1, SF1	119	91	106	111
C 1, SF9	657	131	344	211
unbe., SF1	81	69	84	80
unbe., SF9	321	99	162	127

Tabelle 7.3: Exasol Laufzeiten Einzel(S)- und Mehrnutzer(M)-Durchläufe in Sekunden

Die Graphen in A.1 zeigen jeweils den Median der Queryzeiten und den jeweiligen Min- und Maximalwert. Für die meisten Abfragen befindet sich der Median sehr dicht am Minimalwert, weil, wie schon erwähnt, nur beim ersten Durchlauf deutlich mehr Zeit benötigt wurde.

7.2 Auswertung Apache Ignite

Apache Ignite wurde der offiziellen Dokumentation folgend aufgesetzt [24]. Um ein Schema für die Datenbank benutzen zu können, musste zusätzlich Persistenz konfiguriert werden. Nach Starten des Docker Containers ist die Datenbank zunächst inaktiv und erfordert explizit aktiviert zu werden mittels

```
1 curl 'http://127.0.0.1:8080/ignite?cmd=setstate\&state=ACTIVE \
2       \&ignite.login=ignite\&ignite.password=ignite'.
```

Ignite kann Daten aus CSV-Dateien einlesen, bietet dabei aber keine Konfigurationsmöglichkeiten, was insbesondere bedeutet, dass nur Kommata als Trennzeichen möglich sind. Deshalb mussten in den erzeugten Daten zunächst alle Default-Trennzeichen des DBGEN, |, durch Kommata ersetzt werden. Bei den ersten Benchmark Versuchen konnten die meisten Queries bei einem SF von 1 nicht abgeschlossen werden bei einer Wartezeit von 30 min pro Query. Es war nötig etwas an der Java Heap Size zu ändern und den Garbage Collector zu tunen [25]. Dabei wurde der Java Heap Size auf 10 GB erhöht und auf den G1 Garbage Collector gewechselt. Trotz des Tunings konnten die Queries 13, 16, 19, 20, und 22 nicht beendet werden und wurden jeweils nach mindestens 30 min Wartezeit abgebrochen.

Das einzige getestete Setup mit Apache Ignite war ohne Hardwarebeschränkung und Scale Factor 1. Das ist das Setup, bei dem die beste Performanz zu erwarten ist. Aber die Ergebnisse waren schon sehr schlecht im Vergleich zu Exasol, sodass weitere Tests wenig Sinn ergeben (s. Tabelle 7.6).

Apache Ignite nutzte höchstens 6 Cores und damit nicht mal ein Drittel der möglichen Cores (s. Tabelle 7.4).

Setup	S Avg.	S Max.	M Avg.	M Max.
unbe., SF1	0.99	6,54	1,95	3,02

Tabelle 7.4: Anzahl genutzter Cores

Die RAM-Auslastung war sehr hoch. Über 15 GB wurden belegt bei gerade einmal 1 GB Datenvolumen (s. Tabelle 7.5). Diese Belegung war von Beginn an und war somit konstant.

Setup	S Avg.	S Max.	M Avg.	M Max.
unbe., SF1	14,77	14,79	14,77	14,78

Tabelle 7.5: RAM-Auslastung in GB

Die Laufzeiten betrugen 18,82 min im Einzel- und 20,53 min im Mehrnutzerlauf. Sie sind viel schlechter als Exasol, selbst im Vergleich zum initialen Lauf bei nur einem Core, insbesondere unter Berücksichtigung, dass fünf Queries gar nicht durchgeführt werden konnten. Deshalb wurden keine weiteren Tests, wie oben angesprochen, durchgeführt. Gut ist hingegen, dass bei mehreren Nutzern nur 9,1% mehr Zeit aufgewendet wurde.

Setup	S avg. in s	M avg. in s
unbe., SF1	1136	1204

Tabelle 7.6: Apache Ignite Laufzeiten Einzel(S)- und Mehrnutzer(M)-Durchläufe in Sekunden

Die durchschnittliche Abfrage im Einzeldurchlauf dauerte 36 s (98 s inkl Q18, s. Abb A.2a) und im Multidurchlauf 38 s (95 s inkl. Q18, s. Abb. A.2b). Bei Q18 fällt auf, dass sie deutlich mehr Zeit benötigte, über 15 min, als die nächst langsamste, Q7 mit ca. 2 min. Nur die Queries Q8 und Q17 liegen im selben Zeitbereich wie Exasol.

Insgesamt war es sehr zeitaufwändig Apache Ignite zum Laufen zu bekommen, da viele Kleinigkeiten zu beachten waren, deren Lösung jeweils einer Recherche bedurfte. Die Einzelzeiten, bis auf wenige Ausnahmen, und Gesamtzeit waren sehr schlecht im Vergleich zu Exasol, selbst wenn Exasol auf nur einen Core beschränkt wurde und Ignite nicht. Dazu kommt die sehr hohe RAM-Belegung. Außerdem ist sehr negativ, dass so viele Queries nicht ausgeführt werden konnten bzw. zu lange gebraucht haben. Hier hat vermutlich der Query-Optimizer versagt, denn, wie bei anderen Datenbanken zu sehen, brauchten dieser Queries nicht notwendigerweise sehr viel Zeit.

7.3 Auswertung CrateDB

Für CrateDB wurde der Dokumentation gefolgt zum Aufsetzen mit Docker [8]. Die Dokumentation lieferte schon eine Compose-Datei mit, die zum Aufsetzen reichte. Vor dem Starten des Containers musste noch vm.max_ map_ count angepasst werden via

```
1 sysctl -w vm.max\_ map\_ count=262144.
```

Vor dem Laden der Daten mussten wieder | zu Kommata geändert werden, weil, wie bei Apache Ignite, keine anderen Trennzeichen angegeben werden konnten. Darüber hinaus war es erforderlich, dass Datumsangaben zwischen doppelten Anführungszeichen stehen. Außerdem mussten in der ersten Zeile der Datendateien jeweils die Spaltenüberschriften stehen. Weil ein CrateDB-Container keinen Zugriff auf das lokale Dateisystem hat, mussten die Dateien in das Dateisystem des Containers übertragen werden.

CrateDB konnte die Queries Q2, Q5, Q7, Q9, Q13, Q17, Q19, Q20, und Q21 nicht bearbeiten. Die Abfragen Q2, Q17, und Q20 enthalten Correlated Subqueries für die ein Syntaxfehler geworfen wurde. Q21 benutzt Exists, was ebenso einen Syntaxfehler produzierte. Q4 und Q22 enthalten auch Exists, ließen sich aber leicht umschreiben (wäre offiziell nicht erlaubt für einen TPC-H Benchmark, eine solche Änderung müsste genehmigt werden laut der Standard Specification [54]). Bei den Abfragen Q5, Q7, Q9, Q13, und Q19 wurde abgebrochen nach mindestens 30 min Wartezeit.

Bei CrateDB wurden, wie auch bei Apache Ignite, keine weiteren Tests durchgeführt, weil schon für das leichteste Setup keine guten Ergebnisse abgeliefert wurden. Dazu kam noch

die große Anzahl von nicht durchführbaren Queries, die CrateDB schlecht vergleichbar zu den anderen Datenbank macht.

CrateDB konnte etwa die Hälfte der verfügbaren Cores nutzen wenn nötig, die durchschnittliche Auslastung war gering, wie die Tabelle **??**.

Setup	S Avg.	S Max.	M Avg.	M Max.
unbe., SF1	0,76	10	1,5	9,82

Tabelle 7.7: Anzahl genutzter Cores

Die RAM-Auslastung war genauso hoch wie bei Apache Ignite. Über 15 GB wurden belegt bei gerade einmal 1 GB Datenvolumen (s. Tabelle 7.8). Diese Belegung war auch hier von Beginn an und war somit konstant. CrateDB legt mehrere Replicas pro Node an. Dies erklärt teilweise die hohe Belegung.

Setup	S Avg.	S Max.	M Avg.	M Max.
unbe., SF1	15,85	15,86	15,85	15,86

Tabelle 7.8: RAM-Auslastung in GB

Die Laufzeiten waren verhältnismäßig schlecht. Die Tabelle 7.9 zeigt 8,97 s bzw. 9,27 s für die Durchläufe an. Es konnten nur 12 der 21 Abfragen ausgeführt werden, trotzdem ist die Zeit deutlich langsamer als Exasol. Positiv anzumerken ist, dass beim Mehrnutzerlauf nur wenig Zeit, 3,3%, mehr benötigt wurde.

Setup	S avg. in s	M avg. in s
unbe., SF1	540	555

Tabelle 7.9: CrateDB Laufzeiten Einzel(S)- und Mehrnutzer(M)-Durchläufe in Sekunden

Die Query Q6 sticht hervor mit einer durchschnittlichen Ausführungszeit von 0,488 s im Einzellauf und 0,553 s im Multilauf. Das war deutlich schneller als alle anderen getesteten Datenbanken. Q3 und Q10 hingegen waren sehr langsam. Q3 brauchte 230,689 s bzw. 229,689 s und Q10 188,961 s bzw. 197,11 s (s. A.3).

CrateDB konnte nicht überzeugen. Neun bzw. elf, wenn zwei nicht geändert worden wären, Queries ließen sich nicht durchführen. Die Gesamtzeit war trotzdem langsamer als bei anderen Testkandidaten, außer Ignite. Die durchschnittliche RAM-Auslastung war die höchste aller getesteten Datenbanken.

7.4 Auswertung MariaDB ColumnStore

MariaDB wurde nach Anleitung der Dokumentation mit Docker aufgesetzt [35]. Um Zugriff auf die Datenbank zu erhalten, war noch die Ausführung des folgenden Befehls nötig:

```
1 docker exec benchmark-mariadbsc mariadb \
2 -e "GRANT ALL PRIVILEGES ON *.* TO 'nsc'@'\%'" \
3 "IDENTIFIED BY 'mariadbcs';"
```

Die Queries Q2, Q5, Q17, und Q19 konnten aufgrund der Syntax nicht ausgeführt werden.

Die Tabelle 7.10 zeigt die Ausnutzung der verfügbaren Cores für MariaDB. MariaDB hatet die mit Abstand höchste durchschnittliche Auslastung (bei den unbeschränkten Läufen), ungefähr doppelt so hoch wie die nächste Datenbank, PostgreSQL. Die extremen Maximalwerte sind wohl der Art, wie Docker diese Statistiken ermittelt, geschuldet [**dockerstats**] und stellen nicht die wirklich benutzten Ressourcen dar (22 Cores wären das Limit).

Setup	S Avg.	S Max.	M Avg.	M Max.
C 1, SF1	0,99	4,23	1	1,25
C 1, SF9	1	1,7	1	5,42
unbe., SF1	5,4	16,17	9,42	19,98
unbe., SF9	7,8	47,67	9,45	80,28

Tabelle 7.10: Anzahl genutzter Cores

MariaDB hielt scheinbar den gesamten Datensatz im RAM und musste zeitweise zusätzlichen RAM beanspruchen (vgl. Tabelle 7.11). Damit war die Auslastung besser als bei Ignite und CrateDB, kam aber nicht an Exasol heran.

Setup	S Avg.	S Max.	M Avg.	M Max.
C 1, SF1	2,56	3,55	2,76	3,74
C 1, SF9	9,79	16,52	10,59	18,06
unbe., SF1	2,56	3,48	2,77	3,74
unbe., SF9	10,59	18,21	11,3	20,47

Tabelle 7.11: RAM-Auslastung in GB

Der folgenden Tabelle 7.12 ist zu entnehmen, dass MariaDB CS extrem schnell war für maximale Ressourcen und 1 GB Daten, schneller als alle anderen getesteten Datenbanken. Allerdings lies die Performanz stark nach, sowohl bei steigender Datenmenge als auch bei reduzierten Ressourcen.

Setup	S avg. Rest in s	M avg. Rest in s
C 1, SF1	176	392
C 1, SF9	1865	4100
unbe., SF1	27	33
unbe., SF9	631	1315

Tabelle 7.12: MariaDB CS Laufzeiten Einzel(S)- und Mehrnutzer(M)-Durchläufe in Sekunden

Die Laufzeiten für unbeschränkte Ressourcen und SF 1 waren alle extrem schnell und die meisten waren die schnellsten gemessenen für alle Datenbanken (s. A.4c, A.4d). Die Zeiten für 1 Core und SF 9 (A.4a, A.4b) als anderes Extrem waren sehr langsam im Vergleich zu Exasol. Allerdings ist MariaDB auch die einzige andere getestete Datenbanken, die dieses Szenario überhaupt bewältigen konnte.

MariaDB hatte eine beeindruckende Performanz für die 1 GB Datenmenge bei unbeschränkten Hardwareressourcen. Allerdings fiel die Leistung enttäuschend schnell ab. Auch hier gab es wieder das Problem, dass nicht alle Queries ausgeführt werden konnten.

7.5 Auswertung PostgreSQL

PostgreSQL konnte ohne weiteres als Docker Container gestartet werden. Das Laden der Daten war auch ohne Probleme möglich.

Wie in der Tabelle 7.13 zu sehen ist, gab es bei der Auslastung der Kerne keine Besonderheiten und PostgreSQL lag im Mittelfeld der getesteten Datenbanken.

Setup	S Avg.	S Max.	M Avg.	M Max.
C 1, SF1	0,99	1,14	0,99	1,48
unbe., SF1	2,4	3,53	5,05	6,08
unbe., SF9	2,31	3,1	4,65	6,16

Tabelle 7.13: Anzahl genutzter Cores

Auch bei der RAM-Auslastung lag PostgreSQL im Mittelfeld (s. Tabelle 7.14)

Setup	S Avg.	S Max.	M Avg.	M Max.
C 1, SF1	1,74	1,8	1,75	1,82
unbe., SF1	1,7	1,8	1,75	1,82
unbe., SF9	15,09	16,32	15,52	16,55

Tabelle 7.14: RAM-Auslastung in GB

Es fällt auf, dass PostgreSQL mit den analytischen Datenbanken für SF 1 und unbeschränkte Hardware-Ressourcen mithalten konnte. Auch bei SF 9, unbeschränkt und SF 1, 1 Core lag sie noch in derselben Größenordnung wie MariaDB ColumnStore, insbesondere wenn berücksichtigt wird, dass mehr Queries ausgeführt wurden. Nur für 1 Core und SF 9 waren die Laufzeiten zu hoch und der Durchlauf musste abgebrochen werden.

Setup	S avg. Rest in s	M avg. Rest in s
C 1, SF1	218	471
unbe., SF1	65	82
unbe., SF9	733	928

Tabelle 7.15: PostgreSQL Laufzeiten Einzel(S)- und Mehrnutzer(M)-Durchläufe in Sekunden

PostgreSQLs Zeiten (A.5c, A.5d) für unbeschränkte Hardware-Ressourcen und SF 1 waren nur etwas hinter MariaDB ColumnStore und fast alle schneller als Exasol inkl. einiger Topzeiten. Auch die Zeiten für 1 Core und SF 1 (A.5a, A.5b) waren noch dicht an denen von MariaDB, allerdings mit zwei großen Ausreißern, Q1 und Q18, die jeweils über 8 min gebraucht haben.

PostgreSQL hat positiv überrascht. Sie konnte gute Ergebnisse bei den Laufzeiten und bei der Ressourcenauslastung abliefern. Ein weiterer positiver Faktor ist, dass PostgreSQL keine Probleme mit der Ausführung der Queries hatte.

7.6 Nutzertests

Um zu Überprüfen, wie verständlich der Umgang mit den Skripten ist und ob viel Erfahrung nötig ist für die Einarbeitung, wurden zwei Werkstudenten gebeten eine neue Datenbank in die Skripte zu implementieren. Einer der Studenten, Student A, hatte viel Erfahrung im Aufsetzen von Systemen und dem Umgang mit Linux. Der andere Student, Student B, hatte keine Erfahrung im Aufsetzen von Systemen und nur Basiskenntnisse im Umgang mit Linux. Beide waren vertraut mit SQL. Dem Student A wurde Apache Drill zugewiesen, dem Student B MonetDB.

Jeder hat das Verzeichnis mit den Skripten bekommen, das auch eine kurze Anleitung zum Einfügen einer neuen Datenbank enthält. Am Ende wurden ihnen die folgenden Fragen gestellt:

1. Wie lange hat die Einarbeitung gedauert?
2. Gab es Verständnisprobleme?
3. Wie lange hat das Einbinden einer neuen Datenbank gedauert (abgesehen vom Aufsetzen der Datenbank, also nur Queries und Skripte anpassen)?
4. Gibt es Verbesserungsvorschläge?

Die Antworten waren folgendermaßen:

Student A

1. Einarbeitung war kein großes Problem, ca. 30 min.
2. Verständnis war okay.
3. Noch nicht vollständig geschehen, da Datenbank kompliziert, ca. 1,5 h.
4. Alles automatisieren, damit der Nutzer nichts mehr tun muss.

Student B

1. Zunächste musste sich der Student, laut eigener Aussage, in Folgendes einarbeiten
   - Die Syntax von Shell-Skripten und in Linux Befehle
   - Den Aufbau des Verzeichnisses mit den Skripten und den Aufbau der Skripte selbst
   - Den Umgang mit der MonetDB CLI mclient
   - Den SQL-Dialekt von MonetDB
   - Wie man mit einem Docker Container arbeitet

   Zur weiteren Vorbereitung musste er ebenfalls
   - den MonetDB Container aufsetzen
   - WSL 2 (Windows-Subsystem für Linux) aufsetzen

   Die gesamte Einarbeitung hat 15-20 Stunden gedauert.
2. Anfangs gab es Probleme zu verstehen, was die Skripte eigentlich machen. Die Kurzanleitung war nicht ausreichend und es hätte einer umfangreicheren Erklärung bedurft. Außerdem wären explizite Beispiele besser gewesen als den existierenden Code als Vorlage nehmen zu müssen.

3. Das Anpassen der Queries und Skripte war dagegen besser. MonetDB benutzt fast dieselbe SQL-Syntax wie PostgreSQL, sodass die meisten Queries schon direkt funktioniert haben. Die Skripte erforderten etwas mehr Aufwand, weil sich mehr in den Aufbau eingearbeitet werden musste.

   Die Anpassungen haben 5-10 Stunden benötigt.

4. Für die Zukunft wäre eine bessere Einführung hilfreich, Außerdem würde eine umfangreichere Dokumentation und/oder mehr Kommentare helfen beim Verständnis.

Beide Studenten sind nicht komplett fertig geworden. Student A hatte noch mit Flatfiles zum Einlesen der CSV-Dateien zu kämpfen. Hier muss angemerkt werden, dass Apache Drill eigentlich nicht in die nähere Auswahl der Datenbanken hätte kommen dürfen. Allerdings ist beim ersten Check nicht aufgefallen, dass das Laden der CSV-Dateien so kompliziert ist.

Die unterschiedlichen Vorkenntnisse machen sich deutlich bemerkbar bei der Einarbeitungszeit. Jemand, der sich in Linux auskennt, kann deutlich schneller mit der eigentlichen Arbeit anfangen. Allerdings ist die Einarbeitung nur einmal erforderlich, sodass für weitere Tests dann auch nur die Arbeit zur Integration der Datenbank anfallen würde.

7.7 Zusammenfassung

Es gibt positive und negative Ergebnisse. Insgesamt kann die Arbeit sicherlich als erfolgreich angesehen werden. Aber es gibt noch Verbesserungspotential für die Zukunft.

Die Tabellen 7.18 und 7.19 zeigen jeweils den Median der einzelnen Queryzeiten für Einzelnutzer- und Multinutzerdurchläufe. Das gewählte Szenario ist unbeschränkte Hardwareressourcen und Scale Factor 1, weil nur für dieses für alle Datenbanken Daten vorliegen. Die Spalte „Exasol initial“enthält die Queryzeiten des ersten Laufs, bei dem noch Indizes angelegt wurden (vgl. 7.1), die Spalte „Exasol“enthält den Median der Zeiten der folgenden Durchläufe (ohne den initialen Lauf). Grün eingefärbt ist jeweils die schnellste Zeit, Rot die langsamste. Für manche Queries sind mehrere Einträge Grün bzw. Rot gefärbt, je nachdem ob für Exasol die Zeiten des initialen Laufs oder die Durchschnittszeiten danach

berücksichtigt werden sollen. Die Tabellen 7.16 und 7.17 zeigen jeweils den Median der Gesamtzeiten, wieder mit der Unterscheidung für Exasol.

Positive Ergebnisse

- Es wurde gezeigt, dass die Auswahl von Datenbanken nach bestimmten Anforderungen zwar gute Kandidaten hervorbringt. So konnte sich z.B. MariaDB ColumnStore für unbeschränkte Hardware-Ressourcen und SF 1 hervortun (s. Tabelle 7.16). Aber es hat sich auch herausgestellt, dass die Anforderungen alleine nicht ausreichen. Erst durch das Ausführen von Benchmarks können die guten von den schlechten Datenbanken getrennt werden. So haben manche Datenbanken bei einzelnen Abfragen sehr gute Zeiten (s. Tabellen 7.18, 7.19) oder sind bei kleiner Datenmenge schnell. Sie brechen dann aber bei großen Datenmengen oder limitierten Ressourcen ein (s. Tabellen 7.16, 7.17).

- Insgesamt hat sich Exasol als die beste Datenbank aus den getesteten herausgestellt. Bis auf die Ausnahme des Szenarios unbeschränkte Hardware und Scale Factor 1, war Exasol die schnellste Datenbank (s. Tabelle 7.16) und hatte die geringste RAM-Auslastung (s. Tabelle 7.2).

- Die Nutzertests haben die Anwendbarkeit auf weitere Datenbanken bestätigt. Die erstellten Skripte lassen sich, nach initialer Einarbeitungszeit abhängig von jeweiligen Vorkenntnissen, mit wenig Aufwand anpassen. Da die Durchläufe (für ein Setup) automatisch ausgeführt werden, ist nach der initialen Anpassung nur noch minimaler Aufwand nötig verschiedene Setups zu testen.

Negative Ergebnisse

- Bei mehreren Datenbanken, Apache Ignite und CrateDB, war sehr enttäuschend, dass sie nicht alle Queries aufgrund von Syntaxproblemen ausführen konnten. Hier wäre es für die Zukunft evtl. sinnvoll, die Dokumentation bzgl. SQL-Fähigkeit genauer zu untersuchen. Allerdings ist die Erkenntnis wichtig, dass sich auf die bloße Aussage SQL zu können nicht verlassen werden kann.

- Die Einarbeitungszeit in die Skripte variiert stark mit den Vorkenntnissen des Benutzers. Eine bessere Dokumentation und ausführlichere Erklärungen, wie von einem

der Nutzer angemerkt (vgl. 7.6) würden sicherlich helfen, diese Zeit zu reduzieren. Außerdem wurde mehr Automatisierung gewünscht. Das ließe sich in zukünftiger Weiterentwicklung umsetzen.

Setup	Exasol initial	Exasol	Ignite	CrateDB	MariaDB	PostgreSQL
C 1, SF1	119	91	/	/	176	218
C 1, SF9	657	131	/	/	1865	/
unbe., SF1	81	69	1136	540	27	65
unbe., SF9	321	99	/	/	631	733

Tabelle 7.16: Gegenüberstellung Gesamtzeit Einzelläufe, unbeschränkt, SF 1

Setup	Exasol initial	Exasol	Ignite	CrateDB	MariaDB	PostgreSQL
C 1, SF1	106	111	/	/	392	471
C 1, SF9	344	211	/	/	4100	/
unbe., SF1	84	80	1204	555	33	82
unbe., SF9	162	127	/	/	1315	928

Tabelle 7.17: Gegenüberstellung Gesamtzeit Multiläufe, unbeschränkt, SF 1

Setup	Exasol initial	Exasol	Ignite	CrateDB	MariaDB	PostgreSQL
Q1	3792	3021	47111,5	3982,5	2207,5	12301,5
Q2	3559	3135	7326	/	/	1665
Q3	5347	3912	18109,5	230820,5	768	2017
Q4	3205	2885	8254	14574	1468	1016,5
Q5	2888	2972	19318,5	/	/	1163,5
Q6	3081	2691	15600,5	487	559,5	1907
Q7	3381	2889	65220	/	2091	1611,5
Q8	4179	2745	3583,5	13739	896	980,5
Q9	5201	2640	13499	/	2585	4243,5
Q10	7927	6534	12281	189423	2450	3375,5
Q11	3455	3241	10874,5	8177	445	866,5
Q12	3226	2899	17680,5	19465,5	1062,5	2498
Q13	3502	2621	/	/	895	3270,5
Q14	2820	2958	22057,5	6363,5	622,5	1931,5
Q15	2651	2776	66633	2242	583,5	1973,5
Q16	4537	4079	/	37877,5	848	1989,5
Q17	3264	2990	2999,5	/	/	5922,5
Q18	2864	2834	706956,5	4608	1390	10669,5
Q19	3485	2878	/	/	/	317
Q20	3084	2966	/	/	2072	1153
Q21	3115	2892	97303	/	3116	2820
Q22	2916	2923	/	6138,5	1654,5	1280,5

Tabelle 7.18: Gegenüberstellung Median Einzelzeiten, unbeschränkt, SF 1

Setup	Exa initial	Exa	Ignite	CrateDB	MariaDB	PostgreSQL
Q1	3394,5	3407,5	49464	4215	1685	16513,25
Q2	3854,5	3428	7654,75	/	/	1740,5
Q3	4933	4501,5	20074,75	228508,5	1505	2230,75
Q4	3474,5	3386	8741,5	15656,75	1864,75	1124,75
Q5	3441,5	3360,5	19934	/	/	1296,25
Q6	3497	3357,5	17514,25	545	430,75	2312.25
Q7	3466	3235,5	70666,25	/	2756,25	1953,75
Q8	3099	3259	3779	12897,25	1226,75	1040,75
Q9	3256	3469,5	13819,75	/	3372,75	5210
Q10	7952	8080	13225,75	196835,75	3010,25	4206
Q11	3415,5	3408,5	11965,5	8816	722,5	949,5
Q12	3335,5	3196	17640,75	20043,5	1418,5	3302,5
Q13	3627	3385	/	/	1091	3861
Q14	3167	3376,5	25089	6503,25	860,5	2221,25
Q15	3312,5	3185	66839,5	1953	830,25	2453
Q16	5173,5	4904,5	/	42330,75,5	1145,5	2374,25
Q17	3702	3174,5	3158,75	/	/	7882,25
Q18	3361	3242,5	735439,75	4837,25	1634,5	14848,25
Q19	3070	3230	/	/	/	342,25
Q20	3438	3334	/	/	2832,5	1527,25
Q21	3554,5	3405	110966,25	/	3674	3638,25
Q22	3372	3194	/	6422,5	1354,25	1412,5

Tabelle 7.19: Gegenüberstellung Median Multizeiten, unbeschränkt, SF 1

8 Fazit

Die Ergebnisse der Arbeit können insgesamt als erfolgreich angesehen werden. Es wurde ein Prozess entwickelt und erfolgreich getestet, mit dem Datenbanken für einen bestimmten Anwendungsfall ausgewählt und anschließend getestet werden können. Dabei wurde festgestellt, dass es für bestimmte Szenarien, 1 GB Datenmenge und viele Hardware-Ressourcen, bessere Alternativen, MariaDB ColumnStore und PostgreSQL, für die bisher genutzte Datenbank, Exasol, gibt (vgl. Kapitel 7).

Der Auswahlprozess für Datenbanken (Kapitel 5) mittels vorher definierter Anforderungen (Kapitel 4) bietet noch Verbesserungspotential. Zwar haben die Anforderungen geholfen die riesige Auswahl von Datenbanken auf eine beherrschbare Menge zu reduzieren. Allerdings waren in der ersten Auswahl noch viele Datenbanken, die nicht geeignet waren. Einerseits könnte hier eine größere Anzahl von Anforderungen für eine striktere Auswahl sorgen. Aber andererseits haben sich keine guten Möglichkeiten gefunden, effizient die Menge aller Datenbanken zu filtern, sodass für viele Datenbanken eine einzelne Recherche nötig war. Außerdem haben sich mit Apache Ignite und CrateDB zwei Datenbanken in der finalen Auswahl befunden, die sich als wenig geeignet für den untersuchten Anwendungsfall, repräsentiert durch den TPC-H Benchmark, herausstellten. Eine ausführlichere Untersuchung hätte dies evtl. schon im Vorhinein aufdecken können. Wobei sich wieder die Frage stellt, wie viel Aufwand dafür gerechtfertigt ist, oder ob ein Benchmark nicht schneller wäre. Eine eingehendere Aufarbeitung wäre sicherlich interessant.

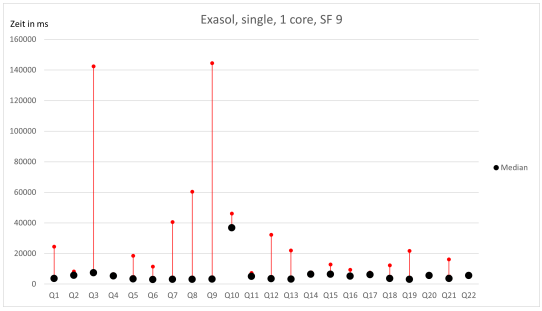
Der ausgewählte Benchmark, TPC-H, hat sich bewährt die zuvor gefundenen und ausgewählten Datenbanken zu evaluieren. So konnten die Stärken einiger Datenbanken bestimmt werden; Exasol war allgemein stark, PostgreSQL und MariaDB ColumnStore waren unter bestimmten Umständen besser als Exasol und sind vollumfänglich kostenfrei und sogar CrateDB und Apache Ignite waren bestimmten Abfragen besser als die anderen Datenbanken. Bei der Auswahl des Benchmarks ist sicherlich Vorsicht geboten, sodass der Benchmark

auch wirklich das Anwendungsszenario darstellt, für den eine Datenbank gefunden werden soll. Z.B. stellt der TPC-H hohe Anforderungen an den Queryoptimizer mit aufwändigen Cross-Joins. Wenn dies in der echten Anwendung nicht vorkommt, könnte das Benchmarken mit dem TPC-H Datenbanken aussortieren, die eigentlich geeignet wären. Auch hier gibt es noch weitere interessante Untersuchungsmöglichkeiten, wie der passendste Benchmark gefunden werden kann oder wie effizient ein eigener zu erstellen ist.

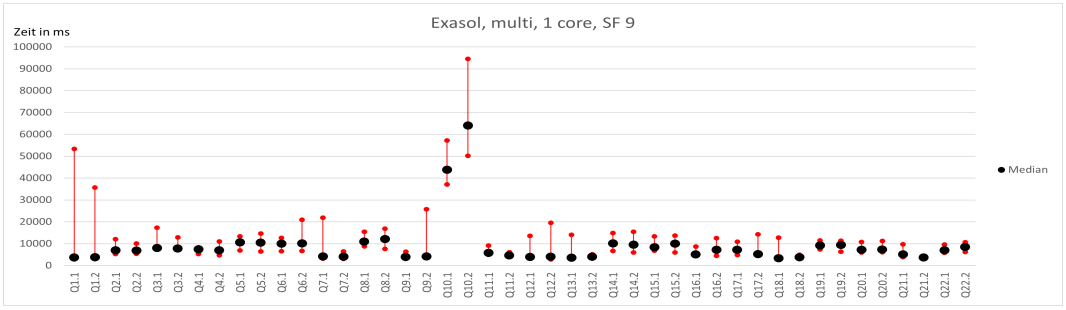
Die erstellten Skripte zur Durchführung haben sich als sehr hilfreich erwiesen. Das Benchmarken einer neuen Datenbank lässt sich mit wenig Aufwand implementieren. Danach wird der Benchmark automatisch durchgeführt. Die Skripte sind flexibel, sodass leicht Anpassungen vorgenommen werden können.

Die Nutzertests konnten bestätigen, dass die Skripte leicht anzupassen sind. Allerdings hat sich auch herausgestellt, dass es noch Verbesserungspotential gibt. Eine bessere Anleitung und Dokumentation würde den Einstieg erleichtern bzw. die erforderlichen Vorkenntnisse stark reduzieren. Dabei würde ebenso ein Refactoring helfen, dass die Skripte etwas aufräumt (s. 7.6). Es hat sich außerdem ergeben, dass das Einbinden einer neuen Datenbank maximal zwei bis drei Arbeitstage dauert und mit entsprechender Erfahrung deutlich schneller ginge. Die Nutzertests haben sich als hilfreich erwiesen, auf Probleme aufmerksam zu machen, die es noch zu verbessern gilt.

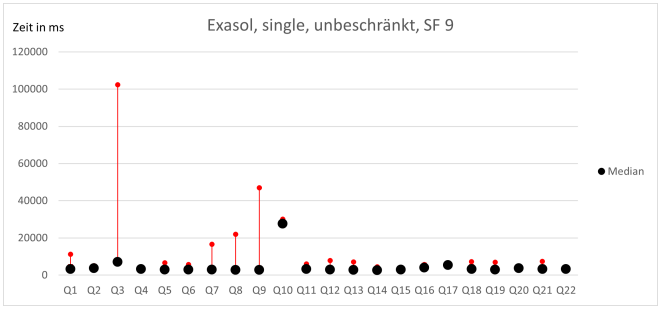
A Appendix



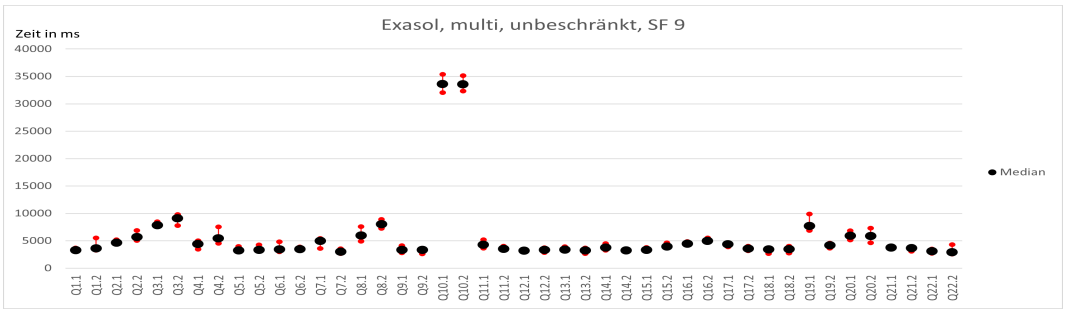
(a) Exasol, single, 1 Core, SF 9



(b) Exasol, multi, 1 Core, SF 9

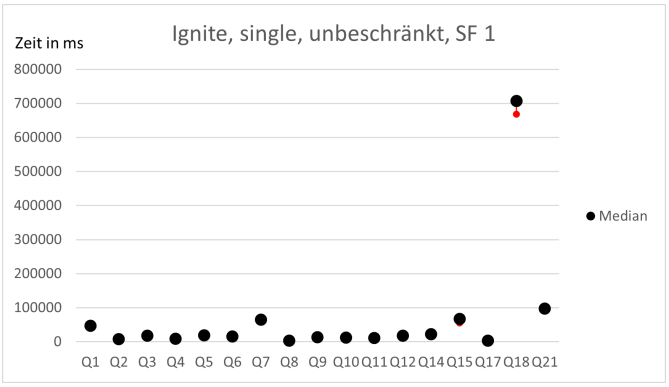


(c) Exasol, single, unbeschränkt, SF 9

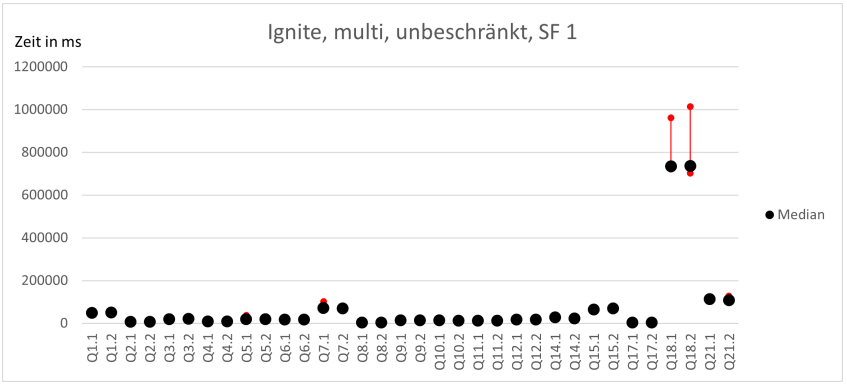


(d) Exasol, multi, unbeschränkt, SF 9

Abbildung A.1: Exasol Laufzeiten

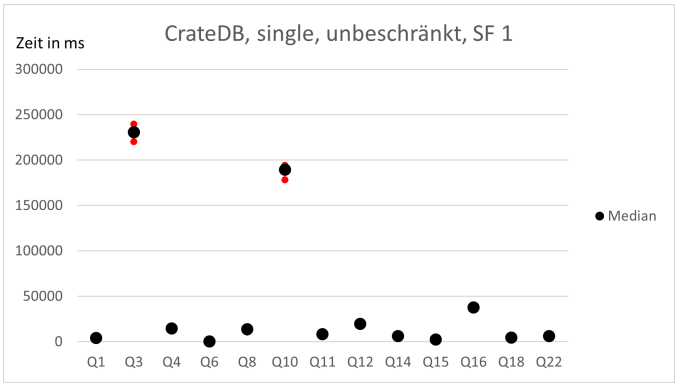


(a) Ignite, single, unbeschränkt, SF 1

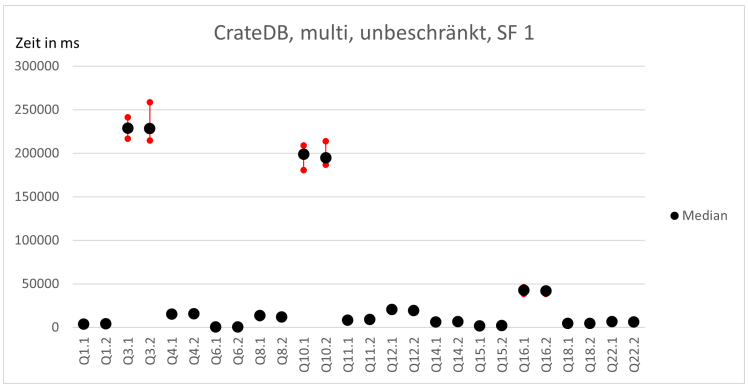


(b) Ignite, multi, unbeschränkt, SF 1

Abbildung A.2: Apache Ignite Laufzeiten

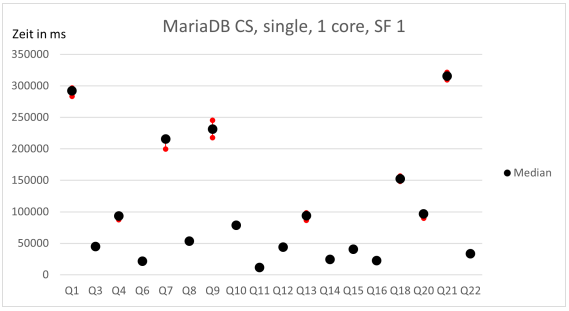


(a) CrateDB, single, unbeschränkt, SF 1

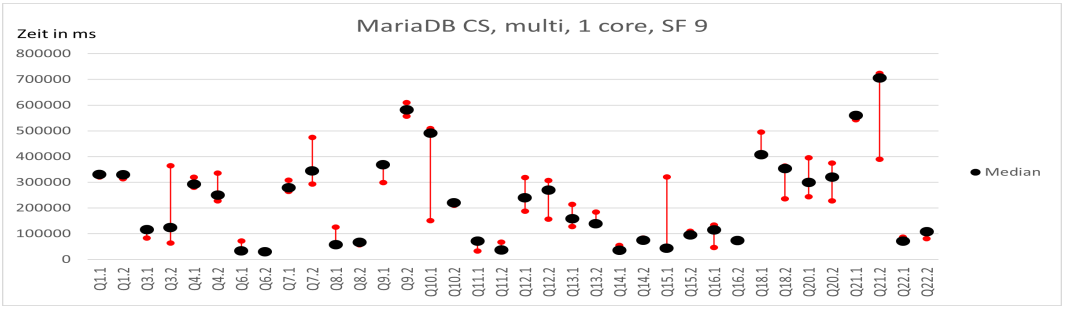


(b) CrateDB, multi, unbeschränkt, SF 1

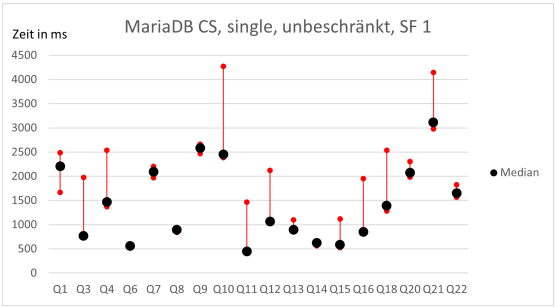
Abbildung A.3: CrateDB Laufzeiten



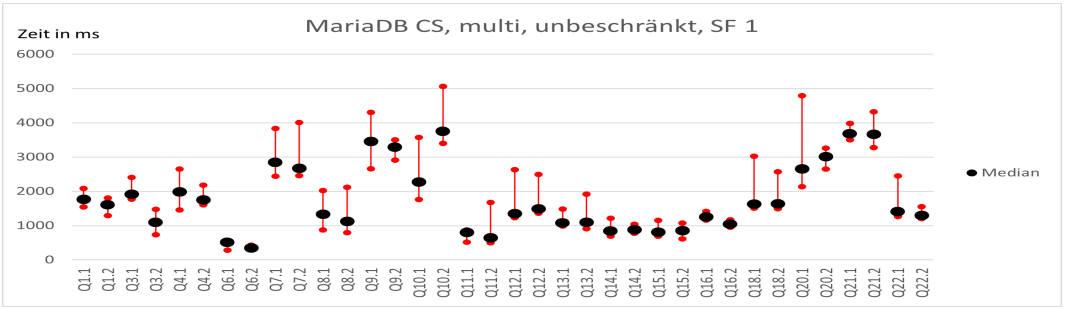
(a) MariaDB CS, single, 1 Core, SF 9



(b) MariaDB CS, multi, 1 Core, SF 9

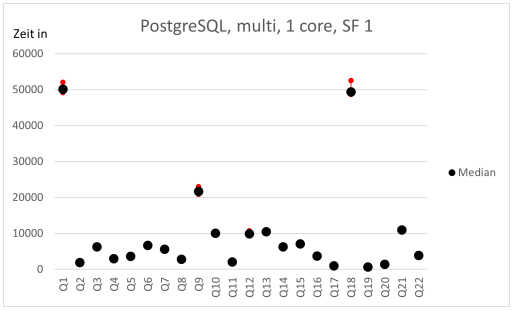


(c) MariaDB CS, single, unbeschränkt, SF 1

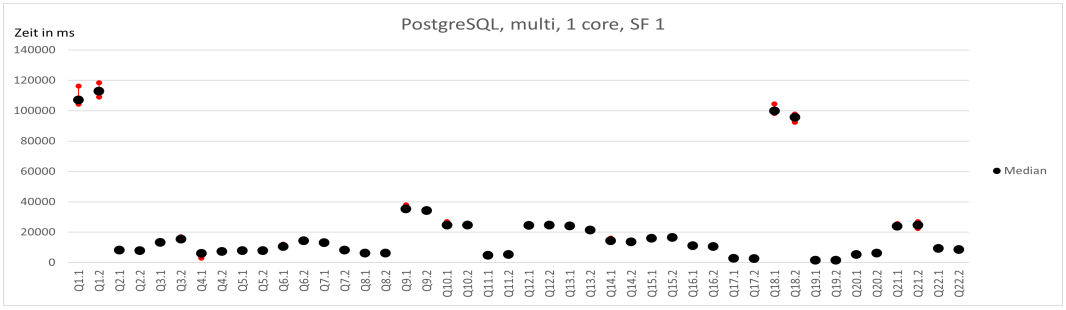


(d) MariaDB CS, multi, unbeschränkt, SF 1

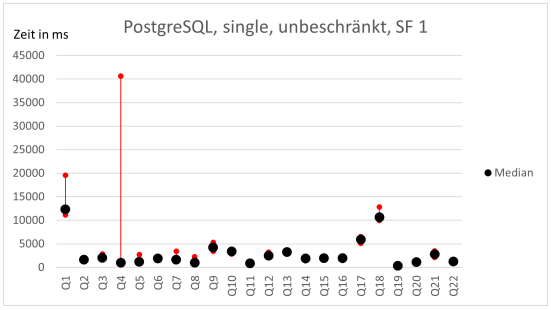
Abbildung A.4: MariaDB CS Laufzeiten



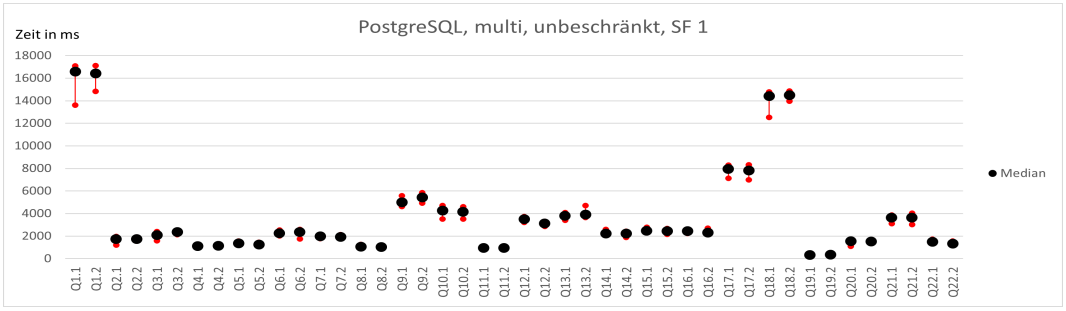
(a) PostgreSQL, single, 1 Core, SF 1



(b) PostgreSQL, multi, 1 Core, SF 1



(c) PostgreSQL, single, unbeschränkt, SF 1



(d) PostgreSQL, multi, unbeschränkt, SF 1

Abbildung A.5: PostgreSQL Laufzeiten

Literatur

[1] Divyakant Agrawal u. a. „Database Scalability, Elasticity, and Autonomy in the Cloud - (Extended Abstract).“ In: Bd. 6587. Jan. 2011, S. 2–15. ISBN: 978-3-642-20148-6. DOI: 10.1007/978-3-642-20149-3_2.

[2] MonetDB B.V. *MonetDB*. https://www.monetdb.org/. accessed 11.01.2022.

[3] David Badalyan und Oleg Borisenko. *Evaluation of TPC-H like workload for Apache Ignite, VoltDB and PostgreSQL*. https://www.isprasopen.ru/2018/docs/Borisenko.pdf. accessed 11.01.2022.

[4] Daniel Benigni. *A Guide to Performance Evaluation of Database Systems*. en. 1984-12-01 1984.

[5] Eric Brewer. „Towards robust distributed systems“. In: Jan. 2000, S. 7. DOI: 10.1145/343477.343502.

[6] Clickhouse. *Clickhouse Benchmark*. https://clickhouse.com/benchmark/dbms/. accessed 11.01.2022.

[7] Crate.io. *Crate.io Homepage*. https://crate.io/. accessed 11.01.2022.

[8] Crate.io. *CrateDB Documentation - RUN CRATEDB ON DOCKER*. https://crate.io/docs/crate/howtos/en/latest/deployment/containers/docker.html. accessed 11.01.2022.

[9] docker. *Docker Homepage*. https://www.docker.com. accessed 11.01.2022.

[10] docker. *Overview of Docker Compose*. https://docs.docker.com/compose/. accessed 11.01.2022.

[11] Markus Dreseler u. a. „Quantifying TPC-H Choke Points and Their Optimizations“. In: *Proc. VLDB Endow.* 13.8 (Apr. 2020), S. 1206–1220. ISSN: 2150-8097. DOI: 10.14778/3389133.3389138. URL: https://doi.org/10.14778/3389133.3389138.

[12] Exasol. *Exasol Homepage*. https://www.exasol.com/de/. accessed 2021.

[13] Exasol. *Exasol Überblick*. https://www.exasol.com/de/uberblick/. accessed 11.01.2022.

[14] The Apache Software Foundation. *Apache Ignite Homepage*. https://ignite.apache.org/. accessed 2021.

[15] J. Gray u. a. „Data cube: a relational aggregation operator generalizing GROUP-BY, CROSS-TAB, and SUB-TOTALS“. In: *Proceedings of the Twelfth International Conference on Data Engineering*. 1996, S. 152–159. DOI: 10.1109/ICDE.1996.492099.

[16] Jim Gray. *Benchmark Handbook: For Database and Transaction Processing Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992. ISBN: 1558601597.

[17] Carnegie Mellon Database Group. *Database of Databases Homepage*. https://dbdb.io/. accessed 11.01.2022.

[18] Theo Haerder und Andreas Reuter. „Principles of Transaction-Oriented Database Recovery“. In: *ACM Comput. Surv.* 15.4 (Dez. 1983), S. 287–317. ISSN: 0360-0300. DOI: 10.1145/289.291. URL: https://doi.org/10.1145/289.291.

[19] Daniel Hieber und Gregor Grambow. „Hybrid Transactional and Analytical Processing Databases: A Systematic Literature Review“. In: Okt. 2020.

[20] Jarmila Horváthová, Martina Mokrišová und Mária Vrábliková. „Benchmarking—A Way of Finding Risk Factors in Business Performance“. In: *Journal of Risk and Financial Management* 14 (Mai 2021), S. 221. DOI: 10.3390/jrfm14050221.

[21] Dongxu Huang u. a. „TiDB: A Raft-Based HTAP Database“. In: *Proc. VLDB Endow.* 13.12 (Aug. 2020), S. 3072–3084. ISSN: 2150-8097. DOI: 10.14778/3415478.3415535. URL: https://doi.org/10.14778/3415478.3415535.

[22] Julian Hyde. *sqlline*. https://github.com/julianhyde/sqlline. accessed 11.01.2022.

[23] „IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries“. In: *IEEE Std 610* (1991), S. 1–217. DOI: 10.1109/IEEESTD.1991.106963.

[24] Apache Ignite. *Apache Ignite Documentation*. https://ignite.apache.org/docs/latest/. accessed 11.01.2022.

[25] Apache Ignite. *Apache Ignite Documentation - Memory and JVM Tuning*. https://ignite.apache.org/docs/latest/perf-and-troubleshooting/memory-tuning. accessed 11.01.2022.

[26] Open Source Initiative. *The Open Source Definition*. https://opensource.org/osd. accessed 11.01.2022.

[27] solid IT GmbH. *DB-Engines Homepage*. https://db-engines.com/en/. accessed 11.01.2022.

[28] solid IT GmbH. *DB-Engines Ranking*. https://db-engines.com/en/ranking. accessed 11.01.2022.

[29] solid IT GmbH. *System Properties Comparison EXASOL vs. Ignite*. https://db-engines.com/en/system/EXASOL3BIgnite. accessed 11.01.2022.

[30] Ralph Kimball und Margy Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. 2nd. USA: John Wiley & Sons, Inc., 2002. ISBN: 0471200247.

[31] Veikko Krypczyk und Elena Bochkor. *Handbuch für Softwareentwickler*. Bonn: Rheinwerk Verlag, 2021.

[32] Kyligence. *Star Schema Benchmark on Apache Kylin*. https://github.com/Kyligence/ssb-kylin. accessed 11.01.2022.

[33] Peter Loos u. a. „In-memory Databases in Business Information Systems“. In: *Business & Information Systems Engineering* 3 (Dez. 2011), S. 389–395. DOI: 10.1007/s12599-011-0188-y.

[34] MariaDB. *MariaDB ColumnStore*. https://mariadb.com/kb/en/mariadb-columnstore/. accessed 11.01.2022.

[35] MariaDB. *Running MariaDB ColumnStore Docker containers on Linux, Windows and MacOS*. https://mariadb.com/kb/en/running-mariadb-columnstore-docker-containers-on-linux-windows-and-macos/. accessed 11.01.2022.

[36] Peter M. Mell und Timothy Grance. *SP 800-145. The NIST Definition of Cloud Computing*. Techn. Ber. Gaithersburg, MD, USA, 2011.

[37] Raghunath Othayoth Nambiar und Meikel Poess. „The Making of TPC-DS“. In: *Proceedings of the 32nd International Conference on Very Large Data Bases*. VLDB '06. Seoul, Korea: VLDB Endowment, 2006, S. 1049–1058.

[38] Pat O'neil, Betty O'neil und Xuedong Chen. „The Star Schema Benchmark (SSB)“. In: (Jan. 2009).

[39] Patrick E. O'Neil. „The Set Query Benchmark". In: *The Benchmark Handbook*. 1991.

[40] Nigel Pendse und Richard Creed. *What is OLAP?* http://www.bi-verdict.com/fileadmin/FreeAnalyses/fasmi.htm. accessed via https://archive.org/web/ Stand 11.4.2010.

[41] Hasso Plattner. „The Impact of Columnar In-Memory Databases on Enterprise Systems: Implications of Eliminating Transaction-Maintained Aggregates". In: *Proc. VLDB Endow.* 7.13 (Aug. 2014), S. 1722–1729. ISSN: 2150-8097. DOI: 10.14778/2733004.2733074. URL: https://doi.org/10.14778/2733004.2733074.

[42] Meikel Poess und Chris Floyd. „New TPC Benchmarks for Decision Support and Web Commerce". In: *SIGMOD Rec.* 29.4 (Dez. 2000), S. 64–71. ISSN: 0163-5808. DOI: 10.1145/369275.369291. URL: https://doi.org/10.1145/369275.369291.

[43] Meikel Poess, Tilmann Rabl und Hans-Arno Jacobsen. *Analysis of TPC-DS - the First Standard Benchmark for SQL-Based Big Data Systems*. 2017.

[44] Columbus Salley und E. F. Codd. „Providing OLAP to User-Analysts: An IT Mandate". In: 1998.

[45] Daniel Seybold. „An automation-based approach for reproducible evaluations of distributed DBMS on elastic infrastructures". Diss. Universität Ulm, 2021.

[46] Kim Shanley. *Origins of the TPC and the first 10 years*. http://tpc.org/information/about/history5.asp. February, 1998, accessed 11.01.2022.

[47] SingleStore. *SingleStore Blog Benchmark*. https://www.singlestore.com/blog/memsql-tpc-benchmarks/. accessed 11.01.2022.

[48] Shaun Snapp. *The Problems with the Strange Lenovo HANA Benchmark*. https://www.brightworkresearch.com/the-problems-with-the-strange-lenovo-hana-benchmark/. accessed 11.01.2022.

[49] Michael Stonebraker. „A New Direction for TPC?" In: *TPCTC*. 2009.

[50] Mike Stonebraker u. a. „C-Store: A Column-Oriented DBMS". In: *Proceedings of the 31st International Conference on Very Large Data Bases*. VLDB '05. Trondheim, Norway: VLDB Endowment, 2005, S. 553–564. ISBN: 1595931546.

[51] A Thanopoulou, Paulo Carreira und Helena Galhardas. „Benchmarking with TPC-H on Off-the-Shelf Hardware: An Experiments Report". In: Jan. 2012.

[52] TPC. *Benchmark Status April 2000*. http://www.tpc.org/reports/status/bs-2000-04.asp. accessed 11.01.2022.

[53] TPC. *TPC Benchmark Status June 2012*. http://www.tpc.org/reports/status/bs-2012-06.asp. accessed 11.01.2022.

[54] TPC. *TPC Downloads*. http://tpc.org/tpc_documents_current_versions/current_specifications5.asp. accessed 11.01.2022.

[55] TPC. *TPC Homepage*. http://tpc.org/. accessed 11.01.2022.

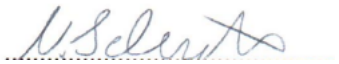
[56] TPC. *TPC-DS V2 Top Performance Results*. http://tpc.org/tpcds/results/tpcds_perf_results5.asp?resulttype=all&version=2. accessed 11.01.2022.

[57] Inc. Wikimedia Foundation. *List of column-oriented DBMSes*. https://en.wikipedia.org/wiki/List_of_column-oriented_DBMSes. accessed 11.01.2022.

Name: Nikolai Schuster Matrikelnummer: 668463

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.



Ulm, den 14.02.2022 .....

Nikolai Schuster