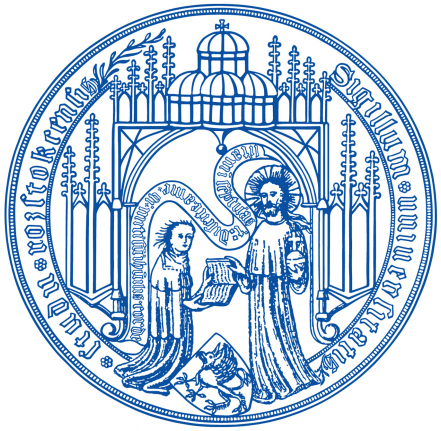
EIGNUNG VON R UND SQL FÜR ALGORITHMEN DES MASCHINELLEN LERNENS

Bachelorarbeit

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik



vorgelegt von

Frank Röger
Mat.-Nr. 210204261
geboren am 04. Januar 1979 in Braunschweig

eingereicht am 19.08.2019

Betreuer

Hannes Grunert
Lehrstuhl für Datenbank- und Informationssysteme
Universität Rostock

Erstgutachter

Dr.-Ing. Holger Meyer
Lehrstuhl für Datenbank- und Informationssysteme
Universität Rostock

Zweitgutachter

Dr.-Ing. Frank Krüger
Institut für Nachrichtentechnik

Universität Rostock

Abstract

Datenbankmanagementsysteme können große Datenmengen effizient speichern und auf Anforderung strukturiert zur Verfügung zu stellen, standardmäßig mit SQL. Algorithmen des maschinellen Lernens benötigen häufig diese großen Datenmengen, werden allerdings in R oder MATLAB programmiert. R besitzt durch seine Bibliothekenlandschaft fast unerschöpfliche Möglichkeiten bei der Durchführung von statistischen Operationen und ist allgemein sehr leistungsfähig, wenn die Datenmenge überschaubar ist. Bei großen Datenmengen müssen allerdings Leistungseinbußen hingenommen werden. Diese Bachelorarbeit erarbeitet Kopplungsmöglichkeiten zwischen R und SQL. Auf dieser Grundlage wird zudem untersucht, welche Auswirkungen der verfügbare Hauptspeicher und die Art des Massenspeichers im Zusammenhang mit den Kopplungsmöglichkeiten und unterschiedlichen Datengrößen haben. Um den Zusammenhang herzustellen, werden in dieser Arbeit verschiedene Testszenarien eingesetzt, um Einlese- und Verarbeitungszeiten untereinander vergleichen zu können.

Database management systems are able to efficiently store and structure large amounts of data on demand, using SQL as standard. Machine learning algorithms often require these large amounts of data, but are programmed in R or MATLAB. R has almost inexhaustible ways to conduct statistical operations through its library landscape and is generally very powerful when the amount of data is manageable. For large amounts of data, however, performance losses must be accepted. The objective of this bachelor thesis is to develop coupling possibilities between R and SQL. Furthermore, effects of the available main memory and the type of mass memory are analysed in combination with coupling possibilities and different dataset sizes. Read-in and processing times are compared using different test scenarios.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listingverzeichnis

Abkürzungsverzeichnis

CSV	Comma-Separated Values
DB	Datenbank
DBMS	Datenbankmanagementsystem
ODBC	Open Database Connectivity
ORE	Oracle R Enterprise
PgSQL	PostgreSQL
SPI	Server Programming Interface
SQL	Structured Query Language
UDF	User Defined Function

1. Einleitung

In der heutigen Zeit sind smarte Geräte nicht mehr wegzudenken. Die meisten dieser Geräte verfügen über zahlreiche Sensoren und Funktionen, um Daten (z.B. Umweltbedingungen, Nutzereingaben, etc.) aufzuzeichnen. Die Menge der dadurch entstandenen Daten können in der Regel nicht (mehr) per Hand ausgewertet werden. Stattdessen setzt man auf intelligente Systeme, die Algorithmen des maschinellen Lernens anwenden. In diesem Zusammenhang trifft man auf zwei typische Problemstellungen, für die es jeweils getrennte Lösungen gibt. Maschinelles Lernen erfordert zum einen eine Datenmenge, aus der gelernt werden kann, um aus den gewonnen Kenntnissen eine Entscheidung treffen zu können. Um eine möglichst präzise Entscheidung treffen zu können, sollte deshalb eine möglichst große Datenmenge zum Lernen vorhanden sein, damit auch seltenere (Entscheidungs-) Fälle oder nicht klar definierte Grenzfälle abgedeckt werden können. Für die Speicherung großer Datenmengen haben sich Datenbanken entwickelt, allen voran die relationalen Datenbankmanagementsysteme (DBMS). Diese sind in der Lage, große Datenmengen strukturiert zu speichern. Zudem bieten sie mithilfe von speziellen Abfragesprachen (in der Regel SQL) die Möglichkeit, selektiv auf bestimmte Datensätze zuzugreifen und in Form von tabellenähnlichen Strukturen zur Verfügung zu stellen. Datenbanken bieten allerdings nur eine eingeschränkte Auswahl an statistischen Methoden. Diese sind allerdings nötig, um andererseits statistische Operationen zur effizienten Verarbeitung der Daten zu ermöglichen, von denen die Algorithmen abhängig sind. Deshalb wird zur Umsetzung häufig R eingesetzt, denn die Entwicklungsumgebung ermöglicht den optimalen Einsatz von statistischen Methoden. Zudem lässt sich der Funktionsumfang durch das Einbinden von Softwarepaketen, die jeder zur Verfügung stellen kann, erweitern. R hat allerdings einen erheblichen Haken: Bei größeren Datenmengen sinkt die Leistungsfähigkeit erheblich.
Diese beiden Nachteile, die mangelnden statistischen Fähigkeiten von Datenbanken und die Leistungseinbußen bei großen Datenmengen von R, versucht man durch das Koppeln beider Ansätze zu negieren.
Diese Bachelorarbeit befasst sich deshalb mit der Untersuchung von Kopplungsmöglichkeiten von R und SQL. Zunächst werden in Kapitel 2 die Grundlagen von R und SQL erarbeitet, um die weiteren Inhalte dieser Arbeit verstehen zu können. In Kapitel 3 werden die Kopplungsmöglichkeiten evaluiert. Dazu werden zwei Kopplungsrichtungen beachtet: Zum einen aus R heraus auf Datenbanken zuzugreifen und zum anderen in Datenbanken in R geschriebene Algorithmen auszuführen. Die Kopplung wird exemplarisch in PostgreSQL im Detail beschrieben. Andere größere Datenbankmanagementsysteme werden nur vorgestellt, ohne weiter ins Detail zu gehen.
Die eigene Umsetzung erfolgt anschließend in Kapitel 4. Sechs Testszenarien beschäftigen sich mit ausgewählten Kopplungsmöglichkeiten, indem die Einlese- und Berechnungsgeschwindigkeiten, unter Berücksichtigung des verfügbaren Hauptspeichers, die verwendete Art des Massenspeichers und der Größe der zu verarbeitenden Datenmenge, verglichen werden. Kapitel 5 befasst sich darauf aufbauend ausschließlich mit der Interpretation der Testergebnisse und versucht aus diesen entsprechende Schlüsse zu ziehen.
Kapitel 6 fasst die Interpretation und die daraus gewonnen Kenntnisse zusammen, bevor dann in Kapitel 7 ein Ausblick über weitere mögliche Forschungsprojekte, die im Zusammenhang mit dieser Ausarbeitung stehen, gegeben wird.

2. Grundlagen

Im Folgenden werden die Grundlagen von R und SQL vorgestellt. Hierbei werden auf spezifische Aspekte, wie die Datenorganisation und -zugriff, als auch das Einlesen von Datensätzen, eingegangen, die für das Verständnis für die vorliegende Arbeit nötig sind.

2.1. R

Die Interpretersprache R[1] ist eine von S[2] beeinflusste Sprache für statistische Berechnungen und deren graphischer Anzeige VSR19 S2. Sie folgt vorwiegend dem funktionalen Programmierparadigma, obwohl sie selbst als *Multiparadigmensprache* bezeichnet wird, da sie Elemente der dynamischen als auch objektorientierten Programmierung beinhaltet. Unter R versteht man nicht nur die Skriptsprache, sondern auch allgemein die komplette Entwicklungsumgebung, um Skripte ausführen zu können.
Um den Funktionsumfang von R zu erweitern, werden Bibliotheken verwendet, die aus Repositories heruntergeladen und installiert werden können. RStudio[3], eine grafische Entwicklungsumgebung für R, bietet die Möglichkeit, Pakete automatisch herunterzuladen und zu installieren. Die Einbindung erfolgt dann direkt im R-Skript:

```
1  library(data.table)
```

Listing 2.1: library(data.table)

2.1.1. Datenorganisation

Grundsätzlich muss man sich folgenden Satz merken, wenn man R verstehen möchte [Cha14, S170]:

Alles, was existiert, ist ein Objekt und alles, was passiert, ist ein Funktionsaufruf.

Objekte können Datentypen bzw. Datenstrukturen sein. Dies können atomare Datentypen, wie logische, numerische, integer und komplexe Werte, sein, aber auch Zeichenketten und Byte-Repräsentationen sowie leere Mengen zählen dazu.
Vektoren (eindimensional), Matrizen (zweidimensional) und n-dimensionale Arrays bilden homogene Datenstrukturen ab. Listen (eindimensional) als auch Data.Frames (zweidimensional) heterogene Datenstrukturen [VSR19, S13ff, S18ff, S26ff].
Das Data.Frame strukturiert Datensätze ähnlich dem Prinzip der relationalen Datenbanken. Zur Veranschaulichung soll eine Liste mit Filmen, deren Erscheinungsjahr und Bewertung in ein Data.Frame übertragen werden. Die Spalten dieser Liste werden definiert, indem diese als homogene Vektoren angelegt werden. Dazu wird die c()-Funktion verwendet. Diese kombiniert ihre Argumente, die vom gleichen atomaren Datentyp sein müssen, zu einem Vektor [VSR19, S24]. Folgendes Beispiel veranschaulicht die Funktion:

```
1  title <- c("The Fellowship of the ring", "Dirty Dancing",
   "Iron Sky")
2  year <- c(2001, 1987, 2012)
3  rating <- c(8.8, 6.9, 5.9)
```

Listing 2.2: Beispiel 1 "Vektoren"

[1] https://www.r-project.org, zuletzt aufgerufen am 15.08.2019

[2] http://ect.bell-labs.com/sl/S/, zuletzt aufgerufen am 15.08.2019

[3] https://www.rstudio.com/, zuletzt aufgerufen am 15.08.2019

Diese drei Vektoren werden nun zu einem `Data.Frame` zusammengefügt:

```
1  films <- data.frame(title, rating, year)
```

Listing 2.3: Beispiel 2"Data.Frame"

Das fertige `Data.Frame` sieht, wenn man sich dieses anzeigen lässt (siehe Punkt 2.1.2), folgendermaßen aus (die Ziffern am Anfang der Zeilen stellen den Index des Datensatzes dar):

```
                        title rating year
1  The Fellowshop of the Ring    8.8 2001
2               Dirty Dancing    6.9 1987
3                    Iron Sky    5.9 2012
```

Listing 2.4: Ausgabe von Beispiel 2

Einen Schritt weiter geht `Data.Table` (Bibliothek `data.table`), welches das `Data.Frame` durch nützliche Funktionen, wie z.B das in den Benchmarks (siehe Listing 4.7) dieser Abschlussarbeit einfach zu verwendende Gruppieren von Daten, erweitert.

2.1.2. Datenzugriff

Der Zugriff auf die Daten erfolgt über die Verwendung des Objektnamens (hier: `films`). Soll auf nur eine bestimmte Spalte zugegriffen werden, wird an das Objekt die Bezeichnung der Spalte hinzugefügt (z.B. `films$title`). Sind die Spalten nicht benannt, werden diese durchnummeriert und können dann mit `films$V2` (`V2` ist hier die 2. Spalte) abgerufen werden.
Ergebnisse von Berechnung können wiederum in ein Objekt gespeichert werden, um damit im späteren Skriptabschnitt weiterzuarbeiten. Soll zum Beispiel der Mittelwert der Bewertungen errechnet und zwischengespeichert werden, ist der Aufruf folgender Zeile erforderlich:

```
1  mittelwert <- mean(films$rating)
```

Listing 2.5: Mittelwertberechnung in R

Das Ergebnis wird hier in `mittelwert` gespeichert. Zeigt man sich nun den Inhalt an, erhält man folgende Anzeige:

```
1  mittelwert
2  [1] 7.2
```

Listing 2.6: Ausgabe Mittelwertberechnung in R

2.1.3. Dateninput

Die Dateneingabe kann über verschiedene Wege erfolgen:

Manuelle Eingabe der Vektoren

Wie in Listing 2.1 gezeigt, kann man die Vektoren (sowie natürlich auch die anderen Objekte) per Hand befüllen. Diese Methode ist für große Datenmengen allerdings ungeeignet.

Einlesen von CSV-Dateien

Komfortabler geht es durch das Einlesen von `CSV`-Dateien (`Comma-Separated Values`, wobei jedes beliebige Trennzeichen verwendet werden kann). `R` bietet verschiedene Möglichkeiten, die Daten einzulesen.

Die gebräuchlichste ist die Funktion `read.csv()`:

```
1  dateiinhalt <- read.csv(file="films.csv", header = TRUE, sep = ";",
   quote = "")
```

Listing 2.7: Einlesen von CSV-Dateien in R

Hier kann man neben der obligatorischen Angabe der Datei auch definieren, ob die erste Zeile den Tabellenkopf (`Header`) darstellt (womit man die Spalten benennen kann), welches Trennzeichen verwendet werden soll; und welches Zeichen Zeichenketten umschließt (im Beispiel wurde keines angegeben) [Rct19, S10].
Neben `read.csv()` existieren noch weitere Möglichkeiten, wie `read.big.matrix()` (Bibliothek `bigmemory`: Für große Matrizen), `read.csv.ffdf()` (Bibliothek `ff`: Auslagern der Daten auf Massenspeichern), `read.csv.sql()` (Bibliothek `sqldf`: Manipulation von `Data.Frames` mit `SQL`-Abfragen).

`Data.Table`[4] bietet ebenfalls die Möglichkeit an, aus `CSV`-Dateien Daten in ein Objekt einzulesen und diese direkt als `Data.Table`-Objekte abzuspeichern:

```
1  dateiinhalt <- fread(file="films.csv", sep = ";")
```

Listing 2.8: fread()

Gegenüber der herkömmlichen Methode (siehe Listing 2.7) besitzt `fread()` einige Vorteile. Zum einen ist der Einleseprozess erheblich kürzer und die Daten im Objekt benötigen weniger Arbeitsspeicher. Aufgrund dieser positiven Eigenschaften wird in den Benchmarks statt `read.csv()` die Funktion `fread()` verwendet.

Einlesen per Datenbankzugriff

Die letzte gängige Möglichkeit Daten einzulesen, stellt der Zugriff mittels Datenbankabfragen, dar. Die Möglichkeiten werden in Punkt 3.1 im Detail betrachtet.

2.1.4. Einschränkungen

`R` hat Probleme mit der Verarbeitung von großen Datenmengen. Die Ausführungsgeschwindigkeit sinkt signifikant, wenn die Datenmengen nicht mehr in den Hauptspeicher passen und die Auslagerungsdatei (bzw. Swappartition) verwendet werden muss [MoE02, S134ff]. Die in dieser Bachelorarbeit erlangten Benchmarkergebnisse belegen diese Aussage, wobei auf diese in der Interpretation näher eingegangen wird.

[4] https://github.com/Rdatatable/data.table/wiki, zuletzt aufgerufen am 15.08.2019

2.2. SQL

Die Structured Query Language, kurz SQL (ausgesprochen S-Q-L oder Sequel) ist eine relationale Datenbankabfragesprache und wird in allen gängigen Datenbanksystemen eingesetzt. Obwohl die generelle Sprachsyntax in jedem Datenbanksystem die gleiche ist, kommen dennoch sogenannte SQL-Dialekte vor, die den allgemeinen Sprachumfang zwar erweitern, zum Teil aber von anderen Datenbanksysteme nicht verstanden werden. Auf dialektspezifische Eigenheiten wird in dieser Arbeit nur in Bezug auf PostgreSQL-Datenbanksysteme[5] (*PostgreSQL* wird mit PgSQL abgekürzt) eingegangen.

Es wird auch darauf hingewiesen, dass sich die folgenden Grundlagen nur auf die Abfrage von Daten beschränken (Data Query Language), da ausschließlich diese für das Verständnis dieser Bachorarbeit nötig sind. Der Vollständigkeit halber sei allerdings erwähnt, dass mit SQL als solches eine Datenmanipulation (Data Manipulation Language), Definition und Änderung des Datenbankschemas (Data Definition Language) und auch eine Rechtverwaltung und Transaktionskontrolle (Data Control Language) möglich wäre.

2.2.1. Datenorganisation

Da SQL, wie eingangs erwähnt, eine relationale Datenbankabfragesprache ist, wird damit auf ein relationales Datenmodell zugegriffen. Die Idee dahinter ist, Datenmengen in strukturierter Tabellenform abzubilden. Die Darstellung erfolgt in Form von Mengenrelationen, deren Datenbasis mindestens eine Tabelle, meist aber aus einer Menge von Tabellen besteht. Oder anders ausgedrückt: Jede Abfrage auf Tabellen generiert eine Ergebnistabelle. Das Grundverständnis von Mengenbegriff ist von zentraler Bedeutung und wird an dieser Stelle vorausgesetzt [SSH18, S19ff].

Eine valide Datenbasis, auf die eine SQL-Abfrage stattfinden könnte, wäre zum Beispiel folgende Tabelle:

title	rating	year	costs	length	id
The Fellowship of the Ring	8.8	2001	93	178	1
Dirty Dancing	6.9	1987	6	100	2
Iron Sky	5.9	2012	10	93	3

Tabelle 2.1: films

Die erste Zeile definiert die Attributsspalten mit ihren Bezeichnungen (hier: title, rating, year, budget, length). Die darauffolgenden Zeilen spiegeln die einzelnen Datensätze mit ihren Attributswerten wider und stellen den Inhalt der Tabelle dar.

[5] https://www.postgresql.org/, zuletzt aufgerufen am 15.08.2019

2.2.2. Datenzugriff

Haupteinsatzzweck von SQL ist die Auswahl (engl. SELECT) bestimmter Informationen aus bestimmten Datensätze. Die Syntax orientiert sich hierbei an der einfachen englischen Befehlsform [SSH18, S31] und ist in ihrer Grundform wie folgt aufgebaut:

```
1  SELECT <Attributsliste>
2  FROM <Relationsliste>
3  [WHERE <Bedingungen>]
4  [GROUP BY <Attributsliste>]
5  [HAVING <Bedingungen>]
6  [ORDER BY (<Attribut [ASC|DESC])+]
```

Listing 2.9: SQL-Syntax

Die FROM-Klausel gibt die Relationen an, auf die zugegriffen werden soll. Auf die obige Tabelle bezogen wäre das die Tabelle films. Die SELECT-Klausel selektiert innerhalb der Tabelle die Attributsspalten, wie z.B. title, rating oder length. Anstatt der Angabe spezifischer Attributsspalten ist auch die Angabe eines Sternes (*) möglich, womit man alle Attributsspalten auswählt. Eine Besonderheit bei der SELECT-Klausel ist, dass hier unter anderem Aggregationsfunktionen (zum Beispiel AVG = Mittelwert) und Skalarfunktionen integriert werden können. Die optionale WHERE-Klausel spezifiziert Selektionsbedingungen, um nur bestimmte Datensätze auszuwählen (z.B. alle Filme mit einem Rating besser als 7.0) [SSH18, S212].
Etwas komplexer wird es mit der GROUP BY-Klausel. Mit dieser ist es möglich, Inhalte von Attributspalten zu gruppieren. Vereinfacht ausgedrückt: Hätte die Tabelle films noch eine weitere Spalte mit der Bezeichnung category, so könnte man die Filme entsprechend ihrer Kategorie miteinander gruppieren. Allerdings muss dann in der SELECT-Teil eine Aggregatfunktiion auf die nicht in der GROUP BY-Klausel vorkommenden Daten erfolgen.
Mit der HAVING-Klausel lässt sich die Gruppierung noch weiter spezifizieren. Zum Beispiel wäre hier die Angabe möglich, alle Filme gruppiert nach Kategorie anzuzeigen (mit GROUP BY), in denen wenigstens fünf Filme vorhanden wären [SSH18, S306f].
Und mit der ORDER BY-Klausel kann man nach Attributsspalten sortieren, wobei zu jeder Attributsspalte bestimmt werden kann, ob diese aufsteigend (ASC) oder absteigend (DESC) sortiert werden soll [SSH18, S308].

Das folgende einfache Beispiel soll die obige Syntax nutzen, um aus der Tabelle films alle Filme auszuwählen, die im Jahr 2000 oder später veröffentlicht wurden. Dazu sollen, die Attributsspalten title und costs der ausgewählten Filme angezeigt werden:

```
1  SELECT title, costs
2  FROM films
3  WHERE year >= 2000;
```

Listing 2.10: SELECT-Beispiel auf Tabelle films

Es ist unerheblich, ob eine Abfrage über eine oder mehrere Zeilen formuliert wird. Das Semikolon signalisiert das Ende der Abfrage.

title	costs
The Fellowship of the Ring	93
Iron Sky	10

Tabelle 2.2: Ausgabe von Listing 2.10

2.2.3. Dateninput

Wie in R gibt es verschiedene Wege, Daten einzulesen:

Manuelle Eingabe der Datensätze

Daten können mittels `INSERT`-Befehl eingefügt werden. Die Syntax folgt dabei in der Regel folgendem Schema:

```
1  INSERT INTO Tabellenname (Spalte1, Spalte2, ...)
2  VALUES (Wert1, Wert2, ...);
```

Listing 2.11: INSERT-Beispiel #1

Wenn Werte für alle Spalten angegeben werden, kann die Angabe der Spaltenbezeichnungen weggelassen werden. Diese Methode ist für große Datenmengen allerdings ungeeignet. Sollen mehrere Tupel gleichzeitig eingefügt werden, ist auch diese Vorgehensweise möglich:

```
1  INSERT INTO Tabellenname (Spalte1, Spalte2, ...)
2  VALUES (Werte), (Werte), ...;
```

Listing 2.12: INSERT-Beispiel #2

Einlesen von CSV-Dateien

Viele Datenbanksysteme bieten das Einlesen von `CSV`-Dateien an. Der Vorgang ist meist immer gleich, nur die Befehle sind meist verschieden. Ein Datensatz stellt eine Zeile dar, die Werte werden durch ein Trennzeichen voneinander getrennt. `PgSQL` ermöglicht das Einlesen mittels folgendem Befehl[6]:

```
1  COPY Tabellenname(Spalte1, Spalte2, ...)
2  FROM 'Dateipfad+Dateiname' DELIMITER 'Trennzeichen' CSV HEADER;
```

Listing 2.13: COPY-Beispiel

Die Angabe des Schlüsselwortes `HEADER` gibt dem Datenbanksystem zu verstehen, dass in der ersten Zeile der Datei die Spaltennamen angegeben sind.

Ausführen von SQL-Dateien

Eine weitere Möglichkeit besteht häufig beim Ausführen von `SQL`-Dateien. Dies sind Textdateien, deren Inhalt aus `SQL`-Befehlen besteht. Vorteil dieser Methode ist es, dass dadurch das Import-Verhalten genau gesteuert werden kann. Zum Beispiel ist es hierbei möglich, eine Datei zu erzeugen, in der durch `SQL`-Befehle zunächst die Tabelle, in der die Datensätze importiert werden sollen, gelöscht und neu erzeugt wird (siehe Listing 4.3).

Transaktionen

Transaktionen werden in Datenbanksystemen verwendet, wenn man möchte, dass eine Gruppe von Teilaktionen in einer oder mehreren Tabellen, als "Paket" (erfolgreich) durchgeführt wird. Schlägt eine dieser Teilaktionen fehl, wird die gesamte Transaktion nicht ausgeführt. Bisher angewendete Teilaktionen werden rückgängig gemacht. Dieses Vorgehen verhindert, dass die Datenbank in einen inkonsistenten Zustand gerät, wenn bei der Durchführung einer Teilaktion ein Problem auftritt [SSH18, S374ff]. Eine Transaktion wird bei `PgSQL` mit `BEGIN;`[7] eingeleitet. Alle `SQL`-Befehle, die anschließend folgen, sind Teil der Transaktion. Abgeschlossen wird die Transaktion mit dem Befehl `COMMIT;` .

[6] https://www.postgresql.org/docs/11/sql-copy.html, zuletzt aufgerufen am 15.08.2019

[7] https://www.postgresql.org/docs/11/sql-begin.html, zuletzt aufgerufen am 15.08.2019

Transaktionen haben allerdings noch einen weiteren Vorteil: Sie beschleunigen den Importvorgang von vielen Datensätzen mit dem Ausführen von SQL-Dateien. Diesen Vorteil erkauft man sich allerdings mit dem Ausnutzen des Arbeitsspeichers. Denn die Daten werden in diesen eingelesen und anschließend verarbeitet. Ist der Arbeitsspeicher voll, müssen Daten ausgelagert werden. In den Benchmarks und der späteren Interpretation werden die Auswirkungen später nur am Rande untersucht.

2.2.4. Einschränkungen

Verwendet man keine In-Memory-Datenbanksystem, hängt die Geschwindigkeit des Datenzugriffs und die Berechnungszeit statistischer Größen vom verwendeten Massenspeicher ab. In der Interpretation der Benchmarkergebnisse wird näher darauf eingegangen.

2.3. Statistische Auswertungsmöglichkeiten im Vergleich

R wird, wie unter Punkt 2.1 beschrieben, für statistische Berechnungen verwendet. Aber auch aktuelle DBMS beherrschen bereits grundlegende Berechnungen der deskriptiven Statistik, wobei dies vom jeweiligen Anbieter abhängt. Die größeren Systeme (PgSQL, OracleDB, Db2) besitzen viele statistische Funktionen. Mittelgroße Systeme, wie MySQL oder SQLite, besitzen dagegen einen Großteil der Fähigkeiten nicht. Tabelle 2.11 soll nach [Sch16, S31] einen Überblick geben, welche Möglichkeiten ausgewählte DBMS mittlerweile besitzen, wobei die Tabelle um die Spalte R erweitert wurde. Ein x in der jeweiligen Spalte bedeutet, dass diese Berechnung von dem jeweiligen System mit einer eingebauten Funktion unterstützt wird.

Berechnung	R	MySQL	SQLite	PgSQL	OracleDB	Db2
Maximalwert	x	x	x	x	x	x
Minimalwert	x	x	x	x	x	x
Mittelwert	x	x	x	x	x	x
Standardabweichung	x	x		x	x	x
Summe	x	x	x	x	x	x
Varianz	x	x		x	x	x
Bestimmtheitsmaß	x			x	x	x
Kovarianz	x		x	x	x	x
Korrelationskoeffizient	x			x	x	x
Lineare Regression	x			x	x	x
y-Achsenabschnitt von lin. Gleichungen	x			x	x	x

Tabelle 2.3: Statistische Funktionen im Vergleich #1

Wie man sehen kann, können mittelgroße Systeme einfache Aggregatsfunktionen noch zur Verfügung stellen, scheitern jedoch bei komplexeren Berechnungen. Die großen Systeme hingegen haben bis hierhin keine Probleme. Da dies jedoch nur einen kleinen Teil des Funktionsumfangs von R darstellt, geht Tabelle 2.12 weiter in die Tiefe, wobei auch hier nur eine Auswahl vorgenommen wurde.

Berechnung	MySQL	SQLite	PgSQL	OracleDB	Db2	Quellen
Median (Zentralwert)	(x)[8]			x	x	Tec19, ibm-1
Fensterfunktionen	x		x	x	x	mys-1, pos-1, ora-1, ibm-2

Tabelle 2.4: Statistische Funktionen im Vergleich #2

Auch diese Funktionen sind, wie in R, durchaus noch mit Bordmitteln der DBMS umzusetzen. Schwierig wird es, wenn es z.B. in Richtung Auswertungsverfahren (Zeitreihen- und Cluster-

[8] Auch wenn keine eingebaute Funktion vorhanden ist, so können die Berechnung über mehr oder weniger komplexe SQL-Abfragen ermöglicht werden. Siehe am Beispiel MySQL und dem Median Sha17.

analyse, etc.) geht. Dies ist zwar auch mit den hier vorgestellten DBMS realisierbar, eleganter und einfacher geht es allerdings mit `R`. Insbesondere die Erstellung von aussagekräftigen Grafiken ist hier ein großer Vorteil. Als Vergleichsbeispiel sei hier [mys-2] und [sha15] zu nennen, in denen sehr gut zu erkennen ist, wie einfach in `R` eine Verteilungsfunktion umgesetzt werden kann.

3. Kopplungsmöglichkeiten

In Kapitel 2 wurden R und SQL getrennt voneinander betrachtet. In diesem Kapitel geht es darum, beide Technologien miteinander zu verbinden, um die Schwächen mit jeweils der anderen Technologie auszugleichen. In diesem Zusammenhang müssen zwei Vorgehensweisen betrachtet werden. Zum einen, welche Möglichkeiten es in R gibt, auf große (strukturierte) Datenmengen aus Datenbanken zuzugreifen und zum anderen welche Möglichkeiten es in R gibt, auf Datenbankebene zu integrieren.

3.1. SQL in R

Durch die Einbindung weiterer Funktionen durch Bibliotheken lässt sich auch die Anbindung zu Datenbanken realisieren. Die nachfolgend aufgeführten Pakete[9] sind auf ein jeweiliges DBMS spezialisiert:

Bibliothek	Abhängigkeit	DBMS
RMySQL	DBI	MySQL
RPostgreSQL	DBI	PostgreSQL
ROracle	DBI	Oracle
RSQLite		SQLite
RMariaDB		MariaDB
ODBC		alle ODBC-fähigen DBMS

Tabelle 3.1: Datenbank-Pakete in R

Wie man sehen kann, werden zahlreiche DBMS nativ unterstützt. Für die anderen DBMS, wie z.B. Db2 oder Microsoft SQL Server, erfolgt die Anbindung über ODBC (Open Database Connectivity). Die Spalte *Abhängigkeit* muss an dieser Stelle genauer betrachtet werden.

DBI ist ein Datenbankinterface [DBI-1], das virtuelle Klassen zur Verfügung stellt, um auf Datenbanken zuzugreifen. Vorteil dieser Bibliothek ist, dass der R-Anwender sich nicht um spezifische Methoden der einzelnen Datenbank-Bibliotheken kümmern muss, sondern diese immer einheitlich sind. Alle Bibliotheken, mit Abhängigkeit zu DBI, setzen voraus, dass DBI in der Entwicklungsumgebung ebenfalls installiert und eingebunden ist.
Um zum Beispiel auf eine PgSQL-Datenbank zuzugreifen, kann folgendes einfache R-Skript verwendet werden:

```
1  library(RPostgreSQL)
2  library(DBI)
3  con <- DBI::dbConnect(RPostgreSQL::PostgreSQL(),
4                        host = "localhost", port = 5432,
5                        user = "postgres", password = "postgres")
6  df_postgres <- dbGetQuery(con, "SELECT * FROM films")
7  dbDisconnect(con)
```

Listing 3.1: Zugriff auf PgSQL mit R

Zeile 1 und 2 binden die beiden benötigten DB-Bibliotheken ein. Zeile 3 bis 5 definieren zum einen die Verbindungsparameter zur Datenbank und stellen gleichzeitig auch die Verbindung her, die in das Objekt con gespeichert wird. In Zeile 6 erfolgt eine einfache Selektion auf die Tabelle films, deren Ergebnis in das Objekt df_postgres, als Data.Frame, gespeichert wird. Zeile 7 schließt die Verbindung wieder.

[9] https://cran.r-project.org/web/packages/available_packages_by_name.html, zuletzt aufgerufen am 16.08.2019

Mit dbGetQuery sollten nur SELECT-Anfragen ausgeführt werden [DBI-1, Abschnitt *dbGetQuery, Details*]. Grund dafür ist, dass Die Funktion dbGetQuery auf drei von DBI zur Verfügung gestellte Funktionen zugreift, um den Vorgang zu vereinfachen. Diese sind dbSendQuery (Sendet die Abfrage an die Datenbank), dbFetch (Empfängt das Ergebnis der Datenbank) und dbClearResult (gibt die verwendeten Ressourcen wieder frei) [DBI-1, Abschnitt *dbGetQuery, Description*]. Möchte man Daten per INSERT-, UPDATE- oder DELETE-Befehl manipulieren, sollte man die Funktion dbExecute benutzen [DBI-1, Abschnitt *dbExecute*}.

Neben den Funktionen dbConnect, dbGetQuery und dbDisconnect werden noch über 35 weitere Methoden angeboten, mit denen unter anderem neue Tabellen angelegt, verändert oder gelöscht werden können und Datensätze ebenfalls hinzugefügt, verändert oder gelöscht werden können. Auch der Aufruf von benutzerdefinierten Funktionen (User Defined Function, UDF) ist möglich. Der volle Funktionsumfang kann in der offiziellen Dokumentation nachgeschlagen werden [DBI-1].

DBI ist *de fakto* die Standardbibliothek zum Zugriff auf Datenbanken. Erkennen kann man das unter anderem daran, dass die DB-Bibliotheken, die keine Abhängigkeit zu DBI besitzen, trotzdem ihre Methoden von DBI importieren. Zudem setzen viele Datenbankpakete zu namhaften DBMS die Bibliothek DBI voraus. Deshalb wird diese zentrale Bibliothek auch in Testszenario #4 (siehe Kapitel 4) näher betrachtet.

Neben den oben in Tabelle 3.1 aufgeführten Bibliotheken und DBI gibt es noch die Bibliothek RODBC, die als Vorgänger von ODBC angesehen werden kann. Sie wird zwar in Verbindung mit den Bibliotheken ibmdbR (native Verbindung zu Db2-Datenbanken) und mssqlR (native Verbindung zu MS SQL Servern) gebracht (z.B. gibt es von IBM einen offiziellen Guide[10], der auf RODBC aufsetzt), sollte allerdings aufgrund geringerer Performance[11] nicht eingesetzt werden.

3.2. R in SQL

Der umgekehrte Weg stellt das Einbinden von R-Skripten in UDFs der DBMS dar. Obwohl jeder Anbieter dabei seinen eigenen Weg geht, haben diese stets immer eins gemeinsam: Sie benutzen die offizielle R-Entwicklungsumgebung als Grundlage zur Ausführung von R-Skripten. Im Folgenden soll exemplarisch der Ansatz in PgSQL im Detail betrachtet werden. Anschließend werden kurz die Ansätze der anderen DBMS vorgestellt.

PL/R

Im Jahr 2003 begann Joseph E. Conway eine Möglichkeit zu schaffen, R als prozedurale Sprache in PgSQL einzubinden. Dies ist nicht nur für Funktionen, sondern auch für Trigger möglich [Con19, Abschnitt *Overview*]. Neben der Installation per Repository[12] ist auch die Installation durch vorkompilierte Binary-Dateien für verschiedene Betriebssysteme möglich.

[10] https://www.ibm.com/support/knowledgecenter/en/SS6NHC/com.ibm.swg.im.dashdb.doc/connecting/connect_connecting_rstudio.html, zuletzt aufgerufen am 15.08.2019

[11] https://github.com/r-dbi/odbc#benchmarks, zuletzt aufgerufen am 16.08.2019

[12] https://github.com/postgres-plr/plr, zuletzt aufgerufen am 15.08.2019

Nach [Con19, Abschnitt *Functions and Arguments*] und damit auf Grundlage von [Pos-2], sind Funktionen, in denen R-Skripte eingebunden werden sollen, wie folgt aufgebaut:

```
1  CREATE OR REPLACE FUNCTION <Funktionsname>(<Argumententypen>)
   RETURNS <Rückgabetyp> AS '
2     <Funktionskörper>
3  ' LANGUAGE 'plr';
```

Listing 3.2: Funktionsrahmen in PL/R

Als Funktionsnamen wählt man die Bezeichnung der Funktion, die diese am Besten beschreibt. Bei den Argumententypen gibt man, jeweils kommasepariert, die Datentypen an (z.B. für zwei Argumente: `integer`, `integer`). Der Rückgabetyp stellt ebenfalls einen Datentypen dar. Der Funktionskörper kann mit beliebigen R-Code versehen werden. Dieser muss nicht angepasst werden, wenn man diesen aus einem reinen R-Skript übernehmen möchte.

Um auf die übergebenen Argumente zugreifen zu können, werden standardmäßig die Variablenbezeichnungen `arg1...argN` verwendet. Sollen die Variablen selbst benannt werden, so ist dies ab PostgreSQL 8.0 auch möglich. Dazu schreibt man einfach den Argumentationsnamen vor den jeweiligen Datentyp:

```
1  CREATE OR REPLACE FUNCTION r_max (wert1 integer, wert2 integer) RETURNS in-
   teger AS '
2    if (wert1 > wert2){
3      return(wert1)
4    } else {
5      return(wert2)
6    }
7  ' LANGUAGE 'plr';
```

Listing 3.3: PL/R-Beispiel mit Argumentationsnamen

Die Variablen können direkt verwendet werden. Anstelle von `wert1` hätte auch `arg1` stehen können, sofern man die Variable nicht benannt hätte. Interessant wird das Ganze, wenn man Daten aus Tabellen übergibt, um diese in R entsprechend für Berechnungen zu verwenden. Als Grundlage soll wieder die Tabelle `films` dienen.

title	rating	year	costs	length	id
The Fellowship of the Ring	8.8	2001	93	178	1
Dirty Dancing	6.9	1987	6	100	2
Iron Sky	5.9	2012	10	93	3

Tabelle 3.2: films

Definiert man den Grenzwert der Überlänge eines Filmes bei 120 Minuten und möchte nun die Tabelle dafür benutzen, um herauszufinden, ob die jeweiligen Filme Überlänge besitzen, wird zunächst folgende Funktion erstellt:

```
1  CREATE OR REPLACE FUNCTION ueberlaenge (length integer) RETURNS bool AS '
2    if (120 >= length) {
3      return(FALSE)
4    } else {
5      return(TRUE)
6    }
7  ' LANGUAGE 'plr';
```

Listing 3.4: Beispielfunktion "ueberlaenge" in PL/R mit integer-Argumententyp

In Zeile 1 wird in der Argumentenliste `integer` als Datentypen und als Rückgabetyp einen boolschen Ausdruck deklariert. Je nachdem, ob der Film mindestens 120 Minuten Länge besitzt, wird entweder `TRUE` oder `FALSE` zurückgegeben. Aufgerufen wird diese Funktion nun mittels folgendem `SQL`-Aufruf, der dazu das darunterstehende Ergebnis liefert:

```
1  SELECT title, ueberlaenge(length) FROM films;
2  title                     | ueberlaenge
3  --------------------------+-------------
4  The Fellowship of the Ring | t
5  Dirty Dancing              | f
6  Iron Sky                   | f
7  (3 rows)
```

Listing 3.5: Ausgabe von Listing 3.4

Wie man sehen kann, wird einfach die Attributsspalte `length` aus der Tabelle `films` an die Funktion übergeben. Sollen anstatt einzelner Spalten ganze Tabellen übergeben werden, muss der Argumentationsteil entsprechend angepasst werden. `length integer` wird schlicht durch `films` ersetzt:

```
1  CREATE OR REPLACE FUNCTION ueberlaenge(films) RETURNS bool AS '
2    if (120 >= arg1$length) {
3      return(FALSE)
4    } else {
5      return(TRUE)
6    }
7  ' LANGUAGE 'plr';
```

Listing 3.6: Beispielfunktion "ueberlaenge" in PL/R mit films-Argumententyp

Die Übergabe der gesamten Tabelle funktioniert mit folgendem Aufruf [Con19, Abschnitt *Functions and Arguments*]:

```
1  SELECT title, ueberlaenge(films) FROM films;
2  title                     | ueberlaenge
3  --------------------------+-------------
4  The Fellowship of the Ring | t
5  Dirty Dancing              | f
6  Iron Sky                   | f
7  (3 rows)
```

Listing 3.7: Ausgabe von Listing 3.6

Die Funktion `ueberlaenge(films)` wird bei diesem Aufruf auf jeden Datensatz angewendet. Möchte man die Daten der gesamten Tabelle haben und damit arbeiten, kann der Befehl `pg.spi.exec()` verwendet werden, der an beliebiger Stelle im `R`-Code verwendet werden kann [Con19, Abschnitt *Database Access and Support, Normal Support*]:

```
1  CREATE OR REPLACE FUNCTION filmtitle() RETURNS text AS '
2    query <- pg.spi.exec("SELECT * FROM films")
3    return(query$title[1])
4  ' LANGUAGE 'plr';
```

Listing 3.8: Beispiel pg.spi.exec()

Wie bereits in den Grundlagen erklärt, kann man nun auf `query`, welches ein `Data.Frame`-Objekt darstellt, wie ein Array zugreifen. Der Aufruf der Funktion erfolgt wie folgt:

```
1  SELECT * FROM filmtitle();
2           filmtitle
3  --------------------------
4  The Fellowship of the Ring
5  (1 row)
```

Listing 3.9: Ausgabe von Listing 3.8

Aufpassen sollte man bei den übergebenen Datentypen. `PgSQL` führt automatisch eine Serialisierung durch. Einzelne Werte werden in `Vektoren` mit nur einem Element umgewandelt. Wertereihen des gleichen Typs (z.B. eine übergebene Attributsspalte) werden in mehrelementige `Vektoren` umgewandelt, mehrdimensionale `PgSQL`-Arrays des gleichen Typs in `Matrizen` (maximal bis zur 3. Dimension). Zusammengesetzte Datentypen, also ganze Tabellen, werden in `Data.Frames` umgewandelt. Hier ein Überblick über die atomaren Datentypen und wie sie als Argumente serialisiert werden [Con19, *Table 4.1: Function Arguments*]:

PostgreSQL	**R**
boolean	logical
int2, int4	integer
int8, float4, float8, cash, numeric	numeric
bytea	object
alles andere	character

Tabelle 3.3: Übersicht Funktionsargumenttypen in PL/R

Die Rückgabewerte von `R` an `PgSQL` werden ebenfalls serialisiert. Zunächst werden alle Rückgabewerte in den Datentyp `character` umgewandelt. Anschließend werden die umgewandelten Zeichenketten in das Rückgabeformat umgewandelt, sofern möglich. Die endgültige Serialisation folgt dann nach diesen Regeln [Con19, *Table 4.2: Function Result Dimensionality*]:

PgSQL	**R Typ**	**Ergebnis**	**Beispiel**
scalar	Array, Matrix, Vector	Erste Spalte der ersten Reihe	c(1, 2, 3) in R -> 1 in PgSQL
setof scalar	1D Array, > 2D Array, Vector	Mehrzeiliges 1 Spaltenset	array(1:10) in R -> 10 Reihen in PgSQL
scalar	Data.Frame	Textrepräsentation des ersten Spaltenvektors	data.frame(c(1,2,3)) in R -> 'c(1,2,3)‘
setof scalar	2D Array, Matrix, Data.Frame	#Spaltenanzahl >1, Fehler; #Spaltenanzahl == 1; Mehrzeiliges 1 Spaltenset	(as.data.frame(array(1:10, c(2,5))))[, 1] -> 2 Reihen von Skalaren
array	1D Array, > 3D Array, Vector	1D Array	array(1:8, c(2,2,2,2) in R -> {1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8}
array	2D Array, Matrix, Data.Frame	2D Array	array(1:4, c(2,2)) in R -> { {1,3}, {2, 4} }
array	3D Array	3D Array	array(1:8, c(2, 2, 2)) -> { {{1, 5}, {3, 7}}, {{2,6}, {4, 8}} }
composite	1D Array, > 2D Array, Vector	Erste Zeile, Erste Spalte	array(1:8, c(2, 2, 2)) -> 1 Zeile als Skalar
setof composite	1D Array, > 2D Array, Vector	Mehrzeiliges 1 Spaltenset	array(1:8, c(2, 2, 2)) -> 8 Zeilen als Skalar
composite	2D Array, Matrix, Data.Frame	Erste Zeilen, Mehrere Spalten	array(1:4, c(2,2)) -> 1 Zeile mit 2 Spalten
setof composite	2D Array, Matrix, Data.Frame	Mehrere Zeilen, mehrere Spalten	array(1:4, c(2,2)) -> 2 Zeilen mit 2 Spalten

Tabelle 3.4: Übersicht Rückgabetypen in PL/R

PL/R kann nur via PgSQL Server Programming Interface (SPI)[13] auf die Datenbank zugreifen. Das Datenbank-Backend bleibt damit grundsätzlich verwehrt [Con19, Abschnitt *Overview*].

R-Skript in T-SQL

Microsoft führt R-Skripte mittels Sp_execute_external_script aus, indem folgender Funktionsrahmen verwendet wird:

```
1  sp_execute_external_script
2  @language = N'language',
3  @script = N'script'
4    [ , @input_data_1 = N'input_data_1' ]
5    [ , @input_data_1_name = N'input_data_1_name' ]
6    [ , @output_data_1_name = N'output_data_1_name' ]
7    [ , @parallel = 0 | 1 ]
8    [ , @params = N'@parameter_name data_type [ OUT | OUTPUT ] [ ,...n ]' ]
9    [ , @parameter1 = 'value1' [ OUT | OUTPUT ] [ ,...n ] ]
```

Listing 3.10: Funktionsrahmen in T-SQL mit R-Skript

Als Sprache in language muss R angegeben werden. In Zeile 3 innerhalb der einfachen Anführungszeichen befindet sich der Funktionskörper, in dem beliebiger R-Code eingefügt werden darf. Mit Zeile 3 können Daten dem R-Code zur Verfügung gestellt werden, wobei hier jede T-SQL-Abfrage erlaubt ist. Das Ergebnis dieser Abfrage ist stets ein Data.Frame-Objekt. In Zeile 4 kann die Bezeichnung des Eingabedatensatzes definiert werden. Wird dies nicht gemacht, wird die Bezeichnung (InputDataSet) vorgegeben. Zeile 5 kann die Bezeichnung des Rückgabedatensatzes definiert werden (Datentyp: Data.Frame). Wird diese Zeile weggelassen, wird automatisch die Bezeichnung OutputDataSet verwendet. Zeile 7 steuert die parallele Verarbeitung des Skriptes (Standard: ausgeschaltet). Ist diese eingeschaltet, muss WITH RESULT SETS zwingend angegeben werden (als letzte Zeile) [mic-1]. Mit WITH RESULT SETS wird das Ausgabeschema definiert. Und zuletzt können in Zeile 8 und 9 beliebige Ein- und Ausgabeparameter definiert werden [mic-2].

Folgende Datentypen werden grundsätzlich nicht unterstützt, wenn diese für Eingabedatensätze und Parameter benutzt werden (CAST zu einem unterstützten Datentyp kann allerdingsvorgenommen), [mic-2, Abschnitt *Restrictions*]:

- cursor (zeilenweises Verarbeiten auf einem Resultset)
- timestamp
- datetime2, datetimeoffset, time
- sql_variant (variable Datentypen)
- text, image
- xml (semi-strukturierte Daten)
- hierarchyid, geometry, geography
- benutzerdefinierte Datentypen

Zudem werden spezielle Vektoren, wie z.B. +inf, -inf, NaN, nicht unterstützt und werfen bei der Ausführung eine Ausnahmebehandlung.

[13] https://www.postgresql.org/docs/current/spi.html, zuletzt aufgerufen am 15.08.2019

RLANG mit SAP HANA

SAP HANA integriert R mittels RLANG, was im Prinzip der gleiche Weg wie mit PgSQL und dem MS SQL Server ist. Der Prozedurrahmen für RLANG sieht wie folgt aus:

```
1  CREATE PROCEDURE <Prozedurname>( (IN|OUT <Argumentenbezeichnung> <Argumen-
   tentyp>)+)
2  LANGUAGE RLANG AS
3  BEGIN
4     <Funktionskörper>
5  END;
```

Listing 3.11: Funktionsrahmen in RLANG

Mit IN und OUT wird die Richtung des Datenflusses bestimmt. IN sind hier die Daten, die an die Prozedur übergeben werden (übergebene Tabellen werden automatisch in Data.Frame-Objekte umgewandelt), OUT sind die benannte Rückgabedatentyp. Im Funktionskörper kann beliebiger R-Code eingefügt werden. Weist man einer OUT-Variable Daten zu, können diese als Rückgabewerte außerhalb der Prozedur weiterverarbeitet werden. Aufgerufen wird die Prozedur normal mit dem CALL-Befehl [sap-1, S21f].

SAP gibt eine Reihe von Einschränkungen an [sap-1, S6]:

- Als Argumente dürfen nur Tabellen übergeben werden.
- Rückgabewerte dürfen nur als Data.Frame-Objekte übergeben werden, die ebenfalls auf eine Tabelle zeigen müssen.
- Genauso wie der Prozedurename nur Kleinbuchstaben enthalten sollte, trifft dies auch auf Variablen innerhalb der Prozedur zu (inkl. Argumentenbezeichnungen).
- Es muss wenigstens ein OUT-Argumententyp deklariert sein. Zudem müssen diesem Rückgabeargument innerhalb des Funktionskörpers auch Daten zugewiesen worden sein (in Form eines Data.Frame-Objektes).

Diese atomaren Datentypen werden von Tabellen zu `Data.Frame`-Objekten (und zurück) unterstützt [sap-1, S7]:

R-Datentypen	SAP HANA SQL-Datentypen
numeric (integer)	TINYINT
	SMALLINT
	INTEGER
numeric (double)	REAL
	DOUBLE
	FLOAT
	FLOAT(p)
	DECIMAL
	DECIMAL (p,s)
	BIGINT
charZeile acter / factor	VARCHAR
	NVARCHAR
	CLOB
	NCLOB
Data	DATE
DateTime(POSIXct)	TIMESTAMP
	SECONDDATE
Raw	VARBINARY
	BLOB

Tabelle 3.5: Datentypen-Konvertierung in SAP HANA

R^n: R-Prozessor

Oracle geht einen etwas anderen Weg (in Oracle R Enterprise, kurz ORE) und bietet `SQL`-Schnittstellen an [ora-2, S7]. Diese sind:

SQL-Schnittstellen-Funktion	Zweck
rqEval()	Ruft ein R-Skript ohne Eingabecursor auf
rqTableEval()	Ruft ein R-Skript auf und übergibt eine Tabelle
rqRowEval()	Ruft ein R-Skript auf und übergibt immer nur eine Zeile zur gleichen Zeit, bzw. mehrere Zeilen als Stücke
rwGroupEval()	Ruft ein R-Skript mit partitionierten Daten und einer gruppierten Spalte auf
sys.rqScriptCreate	Erstellt ein benanntes R-Skript
sys.rqScriptDrop	Löscht ein benanntes R-Skript

Tabelle 3.6: SQL-Schnittstellen bei Oracle

Mit Ausnahme von `rqEval()` und, wenn man es genau nimmt, `sys.rqScriptCreate` und `sys.rqScriptDrop`, werden Daten mittels Zeiger an das `R`-Skript übergeben [ora-2, S9]. Parameter werden ebenfalls als Zeiger auf skalare numerische Werte oder Zeichenketten übergeben [ora-2, S10]. Der generische Funktionsaufruf sieht dabei so aus [ora-2, S8]:

```
1  <Funktionsname> (
2        <Eingabecursor>,
3        <Parametercursor>,
4        <Definition der Ausgabetabelle>,
5        <Gruppierung>|<Anzahl der Zeilen zur selben Zeit>,
6        'R-Script-Name'
7  )
```

Listing 3.12: Funktionsaufruf bei Oracle

Eine Besonderheit bei Oracle ist die `Definition der Ausgabetabelle`. Damit ist es nicht nur möglich, entsprechende Tabellenstrukturen zu definieren, sondern die Ausgabe im `XML`-Format oder sogar als generiertes `PNG`-Bild zu ermöglichen [ora-2 S8, S15f, S19]. In Listing 3.13 ist ein einfaches Beispiel mit einer Ausgabetabelle mit zwei Spalten zu sehen [ora-2, S11]:

```
1  begin
2    sys.rqScriptCreate('Example1',
3    'function() {
4      ID <- 1:10
5      res <- data.frame(ID = ID, RES = ID / 100)
6      res}');
7  end;
8  /
9  SELECT * FROM TABLE(rqEval(NULL,'SELECT 1 id, 1 res FROM dual',
   'Example1'));
10       ID     RES
11 ------- -------
12        1     .01
13        2     .02
14        3     .03
15        4     .04
16        5     .05
17        6     .06
18        7     .07
19        8     .08
20        9     .09
21       10      .1
22
23 10 rows selected.
```

Listing 3.13: Beispiel einer R-Skript-Funktion bei Oracle

4. Eigenes Konzept

Die Kopplungsmöglichkeiten offenbaren viele Möglichkeiten die statistischen Einschränkungen in SQL aufzuheben. Zudem können R-Skripte direkt in Datenbankumgebungen ausgeführt werden, um mit denen auf große Datenbestände zuzugreifen.

Das hier vorliegende Konzept beschreibt die Verwendung von selbst aufgestellten Testszenarien, die sich mit Zeitmessungen unter ausgewählten Hardwareabhängigkeiten beschäftigten. Dabei sollte der Aspekt der Hardwareabhängigkeit berücksichtigt werden, weshalb vordefinierte Testprofile verwendet wurden.

Anhand dieser Ergebnisse sollte eine anschließende Interpretation und, sofern möglich, die Sinnhaftigkeit und Eignung im Rahmen von maschinellen Algorithmen erörtert werden. Um im statistischen Bereich eine Vergleichsbasis zu R zu besitzen, wurde als Datenbanksystem PgSQL ausgewählt, da dieses in der Lage ist, grundlegende Funktionen wie den Mittelwert, die Standardabweichung und Varianz zur Verfügung zu stellen.

4.1. Testumgebung

Die Durchführung der Benchmarks erfolgte in einer virtuellen Maschine. Das Host-System besaß folgende Konfiguration:

Prozessor:	Intel Core I5-3570K @3,40GHz, 4 Kerne
Hauptplatine:	ASRock Z77 Pro4 (Bios: v1.80)
Arbeitsspeicher:	16 GB DD3 @1600MHz
Massenspeicher:	Samsung SSD 850 EVO (EMT03B6Q) 1TB, Mode: UDMA6
	Seagate BarraCuda ST4000DM004-2CV104 4TB, 5400 U/min, Mode: UDMA6
Betriebssystem:	Ubuntu 19.04 Desktop 64 Bit
Virtualisierungssoftware:	VirtualBox v6.0.6

Tabelle 4.1: Konfiguration des Host-Systems

In Tabelle 4.2 ist die Konfiguration des Gast-Systems zu sehen. Nicht angeführte Angaben, wie z.B. *EFI aktivieren (nur spezielle Gäste)*, waren in den Einstellungen deaktiviert:

Prozessor:	1 Kern, CPU-Begrenzung: 100%
Hauptplatine:	Chipsatz: PIIX3, Paravirtualisierung: KVM Aktivierte Features: IO-APIC, Hardware-Uhr in UTC, VT-x/AMD-V, Nested Paging
Arbeitsspeicher:	2GB/4GB/8GB*
Anzeige:	Grafikspeicher: 16MB, Grafik-Controller: VMSVGA
Massenspeicher:	Controller: SATA (Typ: AHCI, Ports: 1) Host I/O-Cache verwenden aktiviert Festplatte: SATA-Port 0 SSD-Laufwerk aktiviert*
Partitionen:	System- und Datenpartition: 20GB Swappartition: 30GB dynamisch belegt
Netzwerkadapter:	Adapter: Netzwerkbrücke, Typ: Intel PRO/1000 MT Desktop (82540OEM) Promiscuous-Modus: verweigern Kabel verbunden
Betriebssystem:	Ubuntu 18.04.2 LTS Server (Stand: 13.07.19)
Datenbank:	PostgreSQL 11.4 64 Bit (Standardkonfiguration) PL/R per "postgresql-11-plr"-Paket
R:	3.4.4 64Bit

Tabelle 4.2: Konfiguration des Gast-Systems

* je nach ausgewähltem Testprofil

4.2. Testprofile

Basierend auf der Konfiguration des Gast-Systems wurden zum Testen folgende Hardwareprofile verwendet:

#	Arbeitsspeicher (in GB)	Massenspeicher
1	2	SSD
2	2	HDD
3	4	SSD
4	4	HDD
5	8	SSD
6	8	HDD

Tabelle 4.3: Testprofile

Im Folgenden werden die verwendeten Testprofile als `TP`, gefolgt von einer Raute und der Profilnummer (z.B. `TP#1`), bezeichnet.

4.3. Testdaten

Die Testdaten wurden unter Windows 10 mit Visual Studio 2019 Enterprise erstellt. Zur Generierung der Datensätze ist der TPC-H Benchmark[14] (Rev. 2.18.0) verwendet worden. Konkret wurden die Datensätze der Tabelle `customer.tbl` verwendet, die folgende Struktur besaßen[15]:

Bezeichnung	Datentyp	Bemerkung
C_CUSTKEY	integer	Nicht Null, Primärschlüssel
C_NAME	character varying(25)	
C_ADDRESS	character varying(40)	
C_NATIONKEY	integer	
C_PHONE	character varying(15)	
C_ACCTBAL	numeric(8,2)	Für Aggregatberechnungen
C_MKTSEGMENT	character varying(10)	Für "Group by"-Funktion
C_COMMENT	character varying(117)	

Tabelle 4.4: Tabellenspezifikation customer.tbl

Die hier verwendeten Bezeichnungen waren spezifikationsgebunden, die Datentypen spiegelten die Datenfelder in der verwendeten Datenbank wider.

Aufbauend auf der Struktur wurden drei Datenpakete erstellt, die wie folgt bezeichnet werden:

#	Bezeichnung	Anzahl der Datensätze
1	low	2.400.000
2	med	4.800.000
3	high	9.600.000

Tabelle 4.5: Datenpaketgrößen

Die Anzahl der Datensätze wurden so gewählt, dass das

- Datenpaket `low` in den Arbeitsspeicher von `TP#1` passt.
- Datenpaket `med` in den Arbeitsspeicher von `TP#2` passt, aber nicht in `TP#1`.
- Datenpaket `high` in den Arbeitsspeicher von `TP#3` passt, aber nicht in `TP#1` oder `TP#2`.

Im Umkehrschluss wurde `R` als auch `PgSQL` dadurch gezwungen, die Swappartition zu verwenden, um den Einfluss der Massenspeicher zu verstärken, wenn nicht genügend Arbeitsspeicher zur Verfügung stand.

[14] http://www.tpc.org/tpch/, zuletzt aufgerufen am 15.08.2019

[15] http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.18.0.pdf, zuletzt aufgerufen am 15.08.2019

Da die Testdaten bei der Generierung als CSV-Datei vorlagen, wurde eine Konsolenapplikation in C# geschrieben, um diese Daten in eine SQL-Datei umzuwandeln. Dies hatte, wie bereits unter Punkt 2.2.3 erwähnt, den Vorteil, den Importvorgang steuern zu können. Die Umwandlungsfunktion im Detail:

```
1  using System;
2  using System.IO;
3
4  namespace ConvertCSVToSQL
5  {
6    class Program
7    {
8      static void Main(string[] args)
9      {
10       string line;
11
12       using (StreamWriter outputFile = new StreamWriter("create.sql"))
13       {
14         outputFile.WriteLine("DROP TABLE public.customer;");
15         string createLine = "CREATE TABLE public.customer (";
16         createLine += "\"C_CUSTKEY\" integer NOT NULL,";
17         createLine += "\"C_NAME\" character varying(25),";
18         createLine += "\"C_ADDRESS\" character varying(40),";
19         createLine += "\"C_NATIONKEY\" integer,";
20         createLine += "\"C_PHONE\" character varying(15),";
21         createLine += "\"C_ACCTBAL\" numeric(8,2),";
22         createLine += "\"C_MKTSEGMENT\" character varying(10),";
23         createLine += "\"C_COMMENT\" character varying(117),";
24         createLine += "CONSTRAINT customer_pkey PRIMARY KEY
           (\"C_CUSTKEY\"))";
26         createLine += "WITH ( OIDS=FALSE );";
27         outputFile.WriteLine(createLine);
28         outputFile.WriteLine("ALTER TABLE public.customer OWNER TO
           postgres;");
29       }
30
31       using (StreamWriter outputFile = new StreamWriter("customer.sql"))
32       {
33         outputFile.WriteLine("BEGIN;");
34         using (StreamReader inputFile = new StreamReader("customer.tbl"))
35         {
36           while ((line = inputFile.ReadLine()) != null)
37           {
38             string[] dataset = line.Split('|');
39             string outputline = "INSERT into public.customer (";
40             outputline += "\"C_CUSTKEY\", \"C_NAME\", \"C_ADDRESS\",
               \"C_NATIONKEY\", ";
41             outputline += "\"C_PHONE\", \"C_ACCTBAL\", \"C_MKTSEGMENT\",
               \"C_COMMENT\") VALUES (";
42             outputline += dataset[0] + ", \'" + dataset[1] + "\', \'";
43             outputline += dataset[2] + " \', " + dataset[3] + ", \'";
44             outputline += dataset[4] + "\', " + dataset[5] + ", \'";
45             outputline += dataset[6] + "\', \'" + dataset[7] + "\');";
46             outputFile.WriteLine(outputline);
47           }
48         }
49         outputFile.WriteLine("COMMIT;");
50       }
51     }
52   }
53 }
```

Listing 4.1: C#-Code zur Umwandlung von CSV-Daten zu SQL-Datei

Zunächst werden in die Datei `create.sql` die Befehle, die die bestehende Tabelle `public.customer` löschen (Zeile 14), dann eine neue Tabelle mit den angegebenen Datenfeldern anlegen (Zeilen 15-26) und abschließend den Besitzer dieser Tabelle definieren, geschrieben. Ab Zeile 31 wird eine Datei `customer.sql` angelegt und schrittweise aus der geöffneten Datei `customer.tbl` (Zeile 34) Zeile für Zeile `INSERT`-Befehle generiert. Der Rahmen für die Transaktion werden in den Zeilen 33 und 49 gesetzt.

4.4. Testszenarien

Die Testszenarien wurden so gewählt, dass diese zunächst die Möglichkeit bieten, `R` und `SQL` direkt zu vergleichen. Aufbauend auf den so ermittelten Testergebnissen wurden Testszenarien entwickelt, die es ermöglichten, die in dieser Abschlussarbeit erwähnten Kopplungsmöglichkeiten zu testen und anschließend zu bewerten..
Jedes Testszenario wurde in Kombination mit dem Testprofil und dem Datenpaket 10 Mal durchlaufen. Aus diesen Ergebnissen wurde der Mittelwert gebildet. Zur Vereinfachung wurden Testskripte sequenziell aneinandergereiht, um z.B. den zeitaufwändigen Prozess des Dateneinlesens zu verkürzen.

In jedem Testszenario wurden alle Testprofile verwendet, um zu überprüfen, ob es eine Abhängigkeit von der Größe des Arbeitsspeichers, bzw. vom verwendeten Massespeicher gab.

Testszenario #1: Einlesen von Daten

Im ersten Testszenario wurde die Zeit betrachtet, die R benötigt, aus einer CSV-Datei die Daten per fread()-Funktion in das Data.Frame-Objekt einzufügen und die PgSQL benötigt, die Daten in die Tabelle einzufügen, damit diese jeweils für weitere Berechnungen zur Verfügung stehen. In R wurde folgender Code verwendet:

```
1  library(data.table)
2  start <- Sys.time()
3  mydata = fread(file="./tables/customer_low.tbl", sep="|")
4  ende <- Sys.time()
5  timeEinlesen <- difftime(ende, start, units = c("secs"))
6  write(paste(timeEinlesen), file ="./results/times_r_low.csv",
   append = TRUE)
```

Listing 4.2: Testszenario #1: Einlesen von Daten in R

Zunächst wird die Bibliothek data.table geladen, die das gleichnamige Objekt und die nötigen Funktionen bereitstellt. Die Startzeit wird in das Objekt start gespeichert, die Endzeit in das Objekt ende. Die Differenz wird in Sekunden in timeEinlesen gespeichert.
Mittels fread() werden die Daten zeilenweise in das Data.Table-Objekt mydata eingelesen. Die letzte Zeile protokolliert die Zeitdifferenz in der Protokolldatei.

Listing 4.2 wurde per Linux Bash aufgerufen:

```
1  i=1
2  while [ $i -le 10 ]
3  do
4     echo "Durchgang #"$i
5     Rscript ./r-scripts/benchmark"$1""$2".r
6     let i=$i+1
7  done
```

Listing 4.3: Testszenario #1: Bash-Skript zum Aufruf des R-Skriptes

Es wird mindestens ein Parameter erwartet, der das Datenpaket (_low, _med, _high) definiert. Der zweite Parameter, der entweder nicht angegeben wird oder _group sein kann, wird in späteren Testszenarien verwendet und näher beschrieben. In Zeile 5 wird Rscript mit der Angabe des auszuführenden R-Skriptes aufgerufen. Die Zeitmessung erfolgt, wie bereits erwähnt, im R-Skript selbst.

Für PgSQL sah das Einfügen der Daten mittels generierter SQL-Datei (siehe Punkt 4.3, Listing 4.1) folgendermaßen aus:

```
1  BEGIN;
2  INSERT INTO public.customer ("C_CUSTKEY", "C_NAME", "C_ADDRESS",
   "C_NATIONKEY", "C_PHONE", "C_ACCTBAL", "C_MKTSEGMENT", "C_COMMENT")
   VALUES (1, 'Customer#000000001', 'IVhzIApeRb ot,c,E ', 0, '10-989741-2988',
   711.56, 'BUILDING', 'to the even, regular platelets, regular, ironic,
   epitaps nag e');
3  INSERT INTO public customer ...
   ...
4  COMMIT;
```

Listing 4.4: Testszenario #1: Inhalt der verwendeten SQL-Datei

Es ist ersichtlich, dass zunächst eine Transaktion geöffnet wird, alle INSERT-Befehle eingefügt werden und schließlich per COMMIT ausgeführt wird.

Listing 4.4 wurde per Linux Bash aufgerufen (Listing 4.5). In diesem wurde auch die Zeitmessung (Zeile 6, 8 und 9) vorgenommen:

```
1  i=1
2  while [ $i -le 10 ]
3  do
4     echo "Durchgang #"$i
5     PGPASSWORD=postgres psql -U postgres -d postgres -f "./sql-scripts/
      create.sql" -q
6     start=$(date +%s.%N)
7     PGPASSWORD=postgres psql -U postgres -d postgres -f "./sql-scripts/
      customer$1.sql" -q
8     end=$(date +%s.%N)
9     timeEinlesen=$(echo "$end - $start" | bc -l)
10    echo "$timeEinlesen" >> ./results/times_sql"$1$2".csv
11    let i=$i+1
12 done
```

Listing 4.5: Testszenario #1: Bash-Skript zum Einlesen der SQL-Datei mit PgSQL

Es wird mindestens ein Parameter erwartet, der das Datenpaket (`_low`, `_med`, `_high`) definiert. Der zweite Parameter, der entweder nicht angegeben wird oder `_group` sein kann, wird in späteren Testszenarien verwendet und näher beschrieben.
Als erstes wird die Datei `create.sql` ausgeführt, die bei jedem Durchgang die Tabelle, in der die Daten geschrieben werden, löscht und neu anlegt (Zeile 5). In Zeile 7 findet dann das eigentliche Einfügen der Daten statt. Zeile 10 führt eine Protokollierung durch.

Testszenario #2: Statistische Berechnungen

Nach dem Einlesen der Daten, konnten auf das Datenfeld C_ACCTBAL statistische Berechnungen durchgeführt werden. Die Wahl fiel hier auf den Mittelwert, die Standardabweichung und die Varianz, da PgSQL diese als Funktion bereitstellte.
Wollte man die Berechnungen mit Gruppierung durchführen, musste der Linux Bashdatei dem zweite Parameter mit der Zeichenkette "_group" übergeben werden.

Ohne Gruppierung der Daten konnten in R mittels folgender Zeile die Berechnungen durchgeführt werden, wobei mean für Mittelwert, sd für Standardabweichung und var für Varianz eingesetzt werden konnte:

```
1  mw <- mean(mydata$V6)
```

Listing 4.6: Testszenario #2: Mittelwertberechnung in R

$V6 steht hier für den Vektor auf C_ACCTBAL.

Mit Gruppierung ist die Zeile etwas komplexer:

```
1  mw <- mydata[, list(mean=mean(mydata$V6)),by = .(mydata$V7)]
```

Listing 4.7: Testszenario #2: Mittelwertberechnung in R mit Gruppierung

Auf $V6 wird der Mittelwert angewendet, per "by =" wird eine Gruppierung auf $V7 (= C_MKTSEGEMENT) durchgeführt.

Auch hier kann mean durch das Äquivalent für die Standardabweichung und die Varianz ersetzt werden.

Daraus ergibt sich das vollständige R-Skript:

```
1  library(data.table)
2  start <- Sys.time()
3  mydata = fread(file="/home/frank/tables/customer_low.tbl", sep="|")
4  ende <- Sys.time()
5  timeEinlesen <- difftime(ende, start, units = c("secs"))
6  start <- Sys.time()
7  mw <- mean(mydata$V6)
8  ende <- Sys.time()
9  timeMw <- difftime(ende, start, units = c("secs"))
10 start <- Sys.time()
11 std <- sd(mydata$V6)
12 ende <- Sys.time()
13 timeStd <- difftime(ende, start, units = c("secs"))
14 start <- Sys.time()
15 va <- var(mydata$V6)
16 ende <- Sys.time()
17 timeVar <- difftime(ende, start, units = c("secs"))
18 write(paste(timeEinlesen,",",timeMw*1000,",",timeStd*1000,",",
   timeVar*1000), file ="/home/frank/results/times_r_low.csv", append = TRUE)
```

Listing 4.8: Testszenario #2: R-Skript für Zeitmessung

Zeile 18 schreibt alle Zeitdifferenzen in eine CSV-Datei. Die Werte für Mittelwert, Standardabweichung und Varianz werden mit dem Wert 1000 multipliziert, um diese als Millisekunden mit zwei Nachkommastellen darzustellen (z.B. 0,967342324 * 1000 = 967,342324000 -> 967,34 ms),

In PgSQL erfolgten die Berechnungen für die drei statistischen Werte über folgende Zeile (ohne Gruppierung):

```
1  SELECT AVG("C_ACCTBAL") FROM public.customer;
```

Listing 4.9: Testszenario #2: Mittelwertberechnung mit PqSQL

AVG steht hier für die Berechnung des Mittelwertes. Diese Funktion kann durch STDDEV_POP (Standardabweichung) oder VAR_POP (Varianz) ersetzt werden.

Die Berechnungen mit Gruppierungen erfolgten mittels folgender Zeile:

```
1  SELECT "C_MKTSEGMENT", AVG("C_ACCTBAL") FROM public.customer GROUP BY
   "C_MKTSEGMENT";
```

Listing 4.10: Testszenario #2: Mittelwertberechnung mit PgSQL und Gruppierung

AVG kann wieder, je nach gewünschter Berechnung, durch STDDEV_POP und VAR_POP ersetzt werden.

Daraus ergab sich folgendes, erweitertes Bash-Skript:

```
1  i=1
2  while [ $i -le 10 ]
3  do
4    echo "Durchgang #"$i
5    PGPASSWORD=postgres psql -U postgres -d postgres -f "./sql-scripts/
     create.sql" -q
6    start=$(date +%s.%N)
7    PGPASSWORD=postgres psql -U postgres -d postgres -f "./sql-scripts/
     customer$1.sql" -q
8    end=$(date +%s.%N)
9    timeEinlesen=$(echo "$end - $start" | bc -l)
10   start=$(date +%s.%N)
11   PGPASSWORD=postgres psql -U postgres -d postgres -f "./sql-scripts/
     avg$2.sql" -q
12   end=$(date +%s.%N)
13   timeMw=$(echo "($end - $start) * 1000" | bc -l)
14   start=$(date +%s.%N)
15   PGPASSWORD=postgres psql -U postgres -d postgres -f "./sql-scripts/
     std$2.sql" -q
16   end=$(date +%s.%N)
17   timeStd=$(echo "($end - $start) * 1000" | bc -l)
18   start=$(date +%s.%N)
19   PGPASSWORD=postgres psql -U postgres -d postgres -f "./sql-scripts/
     var_pop$2.sql" -q
20   end=$(date +%s.%N)
21   timeVar=$(echo "($end - $start) * 1000" | bc -l)
22   echo "$timeEinlesen,$timeMw,$timeStd,$timeVar" >> ./results/
     times_sql"$1$2".csv
23   let i=$i+1
24 done
```

Listing 4.11: Testszenario #2: Bash-Skript für Zeitmessung in PgSQL

Wie bei den R-Skripten auch werden die Werte für Mittelwert, Standardabweichung und Varianz mit 1000 multipliziert, um die Werte später in Millisekunden angeben zu können.

Um in R die Daten gruppieren zu können, musste der Code aus Listing 4.8 entsprechend angepasst werden:

```
1  library(data.table)
2  start <- Sys.time()
3  mydata = fread(file="./tables/customer_low.tbl", sep="|")
4  ende <- Sys.time()
5  timeEinlesen <- difftime(ende, start, units = c("secs"))
6  start <- Sys.time()
7  mw <- mydata[, list(mean=mean(mydata$V6)),by = .(mydata$V7)]
8  ende <- Sys.time()
9  timeMw <- difftime(ende, start, units = c("secs"))
10 start <- Sys.time()
11 std <- mydata[, list(sd=sd(mydata$V6)),by = .(mydata$V7)]
12 ende <- Sys.time()
13 timeStd <- difftime(ende, start, units = c("secs"))
14 start <- Sys.time()
15 va <- mydata[, list(var=var(mydata$V6)),by = .(mydata$V7)]
16 ende <- Sys.time()
17 timeVar <- difftime(ende, start, units = c("secs"))
18 write(paste(timeEinlesen,",",timeMw*1000,",",timeStd*1000,",",
   timeVar*1000), file ="/home/frank/results/times_r_low_group.csv",
   append = TRUE)
```

Listing 4.12: Testszenario #2: R-Skript für Zeitmessung mit Gruppierung

Im Gegensatz zu Listing 4.8 wurden die Zeilen 7, 11 und 15 verändert. Zunächst wurde eine Liste innerhalb des Data.Table-Objektes mydata erstellt, in der eine Gruppierung (`by = .(mydata$V7)`) und deren Aggregatsfunktion (`mean(mydata$V6)`) ausgeführt wurde. Die Zeitmessung erfolgte sonst analog zu Listing 4.8.

Testszenario #3: PL/R und fread()

Durch Testszenario #1 wurde die Grundlage geschaffen, Vergleiche mit den in Kapitel 3 beschriebenen Kopplungsmöglichkeiten zu ziehen. Im dritten Testszenario wurden deshalb die Einlesezeiten mittels fread() innerhalb von PgSQL-Funktionen untersucht. Abgesehen vom Funktionsrahmen unterschied sich der Code im Funktionskörper nicht von dem aus Listing 4.1 (hier wie auch in den nachfolgenden Listings: Die Funktion für das Datenpaket low):

```
1  library(data.table)
2  start <- Sys.time()
3  mydata = fread(file="/home/frank/tables/customer_low.tbl", sep="|")
4  ende <- Sys.time()
5  timeEinlesen <- difftime(ende, start, units = c("secs"))
6  write(paste(timeEinlesen), file ="/home/frank/results/times_r_low.csv",
   append = TRUE)
```

Listing 4.13: Testszenario #3: R-Skript mit fread() in PL/R

Für jedes Datenpaket (_low, _med, _high) wurde in PgSQL eine entsprechend angepasste Funktionen angelegt. Bei der Erstellung dieser Funktionen war wichtig, die Sprache plr anzugeben. Dies konnte entweder über Tools wie pgAdmin[16] durch die Angabe im Definitionsfeld *Language* oder mittels CREATE erfolgen:

```
1  CREATE OR REPLACE FUNCTION f_testcase3_low() RETURNS integer AS '
1    library(data.table)
2    start <- Sys.time()
3    mydata = fread(file="/home/frank/tables/customer_low.tbl", sep="|")
4    ende <- Sys.time()
5    timeEinlesen <- difftime(ende, start, units = c("secs"))
6    write(paste(timeEinlesen), file ="/home/frank/results/times_r_low.csv",
     append = TRUE)
7  ' LANGUAGE 'plr';
```

Listing 4.14: Testszenario #3: R-Skript mit fread() in PL/R als Befehl

Der Rückgabetyp wurde als integer definiert. Dies spielt allerdings in diesem Testszenario keine Rolle. Der Aufruf dieser Funktionen wurde über eine angepasste SQL-Datei (Listing 4.15) realisiert, die wiederum von einer Bash-Datei aufgerufen wurde (Listing 4.16).

```
1  SELECT * FROM public.f_testcase3_low();
```

Listing 4.15: Testszenario #3: SQL-Aufruf von Listing 4.14

```
1  i=1
2  while [ $i -le 10 ]
3  do
4  echo "Durchgang #"$i
5  PGPASSWORD=postgres psql -U postgres -d postgres -f "./sql-scripts/test
   case3$1$2.sql" -q
6  let i=$i+1
7  done
```

Listing 4.16: Testszenario #3: Bash-Skript zum Aufruf des SQL-Codes in Listing 4.13

Die Argumente in Listing 4.16 verhalten sich dabei analog zu Listing 4.5. Die Zeitmessung erfolgte innerhalb der R-Funktion. Deren Ergebnisse wurden in einer CSV-Datei abgespeichert.

[16] https://www.pgadmin.org/, zuletzt aufgerufen am 15.08.2019

Testszenario #4: PL/R und pg.spi.exec()

Folgte Testszenario #3 dem Ansatz, Daten für statistische Berechnungen zu verwenden, die aus einer CSV-Datei stammen, sollte mit diesem Testszenario die Einlesezeit untersucht werden, wenn die Datenquelle eine SQL-Abfrage innerhalb von PL/R darstellte. Gemessen wurde also wieder die reine Einlesezeit. Die Funktion in PgSQL dazu sah wie folgt aus:

```
1  library(data.table)
2  start <- Sys.time()
3  mydata <- pg.spi.exec("SELECT * FROM customer")
4  ende <- Sys.time()
5  timeEinlesen <- difftime(ende, start, units = c("secs"))
6  write(paste(timeEinlesen), file ="/home/frank/results/times_tc4.csv",
   append = TRUE)
```

Listing 4.17: Testszenario #4: R-Skript mit pg.spi.exec() in PL/R

Aufgerufen wurde diese Funktion mit dem folgenden Listing:

```
1  SELECT * FROM public.f_testcase4();
```

Listing 4.18: Testszenario #4: SQL-Befehl zum Aufruf von Listing 4.17

Dieses wurde mit diesem Bash-Skript aufgerufen:

```
1  i=1
2  while [ $i -le 10 ]
3  do
4  echo "Durchgang #"$i
5  PGPASSWORD=postgres psql -U postgres -d postgres -f "./sql-scripts/
   testcase4.sql" -q
6  let i=$i+1
7  done
```

Listing 4.19: Testszenario #4: Bash-Skript zur Zeitmessung

Testszenario #5: PL/R und statistische Berechnungen

Die bereits in Testszenario #3 bekannten PgSQL-Funktionen wurden erweitert, um in PL/R-Umgebungen statistische Berechnungen durchzuführen. Der Funktionskörper war identisch mit dem Listing 4.8. Zusätzlich wurde für jedes Datenpaket noch eine Funktion mit dem Namenszusatz _group erstellt, deren Funktionscode identisch zu Listing 4.12 war. Das Aufrufen dieser Funktionen geschah analog zu den im Testszenario #3 beschriebenen Vorgehen.

Testszenario #6: R und DBI

In diesem Testszenario wurde in einer reinen R-Umgebung über DBI und RPostgreSQL auf Funktionen der PgSQL-DB zugegriffen. Getestet wurde, wie schnell die Übertragung der Daten mittels dbGetQuery() in ein Data.Frame-Objekt erfolgte. Das dazu benutzte R-Skript sah wie folgt aus:

```
1  library(methods)
2  library(DBI)
3  library(RPostgreSQL)
4  con <- DBI::dbConnect(RPostgreSQL::PostgreSQL(),
5                        host = "192.168.178.43", port = 5432,
6                        user = "postgres", password = "postgres")
7  start <- Sys.time()
8  mydata <- dbGetQuery(con, "SELECT * FROM public.customer");
9  ende <- Sys.time()
10 timeEinlesen <- difftime(ende, start, units = c("secs"))
11 write(paste(timeEinlesen), file ="/home/frank/results/times_tc6.csv",
   append = TRUE)
12 dbDisconnect(con)
```

Listing 4.20: Testszenario #6: R-Skript mit DBI und SQL-Abfrage an PgSQL

Die Bibliothek methods wurde von DBI benötigt. Aufgerufen wurde dieses Skript mit folgendem Bash-Skript:

```
1  i=1
2  while [ $i -le 10 ]
3  do
4  echo "Durchgang #"$i
5  Rscript ./r-scripts/benchmark_tc6.r
6  let i=$i+1
7  done
```

Listing 4.21: Testszenario #6: Bash-Skript zum Aufruf von Listing 4.20

5. Ergebnisse und Interpretation

Dieses Kapitel beschäftigt sich grundlegend mit der Auswertung der einzelnen Testszenarien.

5.1. Ergebnisse

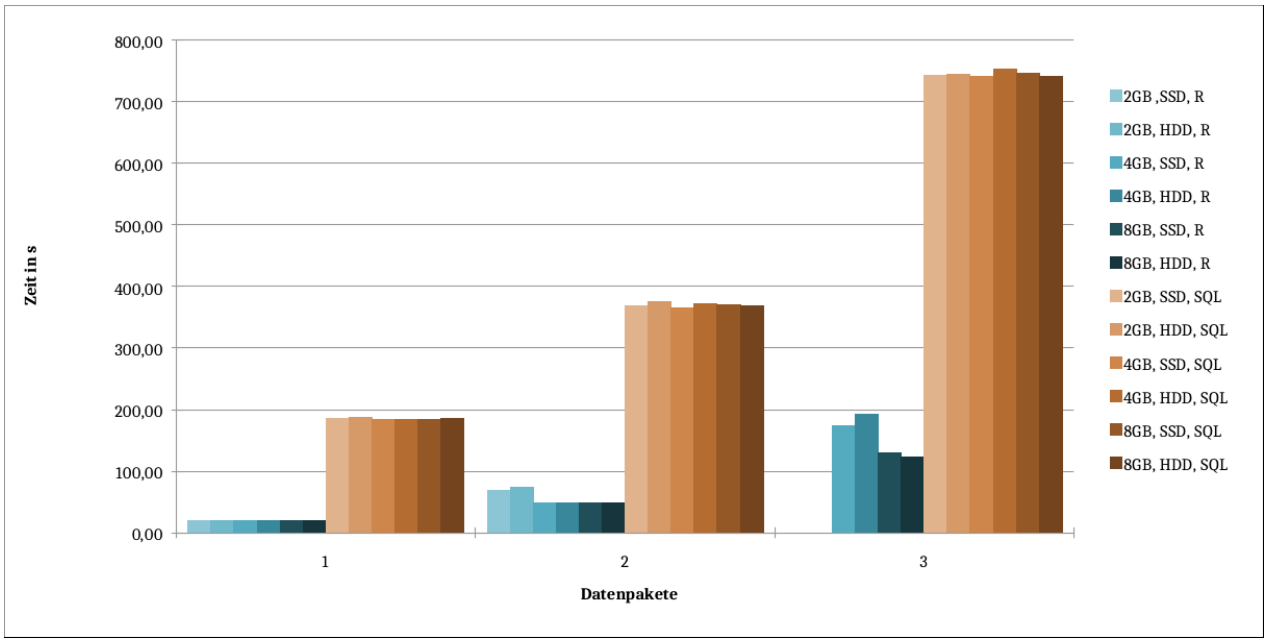
Die Ergebnisse befinden sich in Tabellenform im Anhang. Mit Ausnahme von Testszenario #1, in dem die gemessenen Zeiten in Sekunden angegeben sind, sind alle weiteren Zeitangaben in Millisekunden angegeben. Ist in der jeweiligen Spalte ein Stern (*) eingetragen, so konnte keine Messung durchgeführt werden. Die Interpretation geht näher auf die jeweiligen Umstände ein. Bei manchen Tabellen gibt es zusätzlich eine Prozent-Zeile. Diese zeigt an, wie groß prozentual der Anstieg der beiden verglichenen Spalten ist.

5.2. Interpretation

Ausgewertet werden stets die Mittelwerte der Durchläufe. Auf etwaige Ausreißer in den einzelnen Messungen wird dabei nicht eingegangen. Zu beachten ist, dass die X-Achse programmbedingt bei der Einteilung der Datenpakete die Zahlen 1 bis 3 verwendet, wobei die 1 für das kleine Datenpaket (`low`), die 2 für das mittlere Datenpaket (`med`) und die 3 für das große Datenpaket (`high`) steht.

Testszenario #1: Einlesen von Daten

Da in diesem Testszenario die Einlesezeiten getestet wurden, also wie lange es in `R` mit `fread()` dauert, die Daten aus einer `CSV`-Datei in ein `Data.Frame`-Objekt zu übertragen und im Vergleich dazu, in `SQL` eine innerhalb einer Transaktion eingelesene `SQL`-Datei in eine Tabelle zu schreiben, ist es zunächst interessant, die Zeiten in direktem Vergleich zu betrachten.



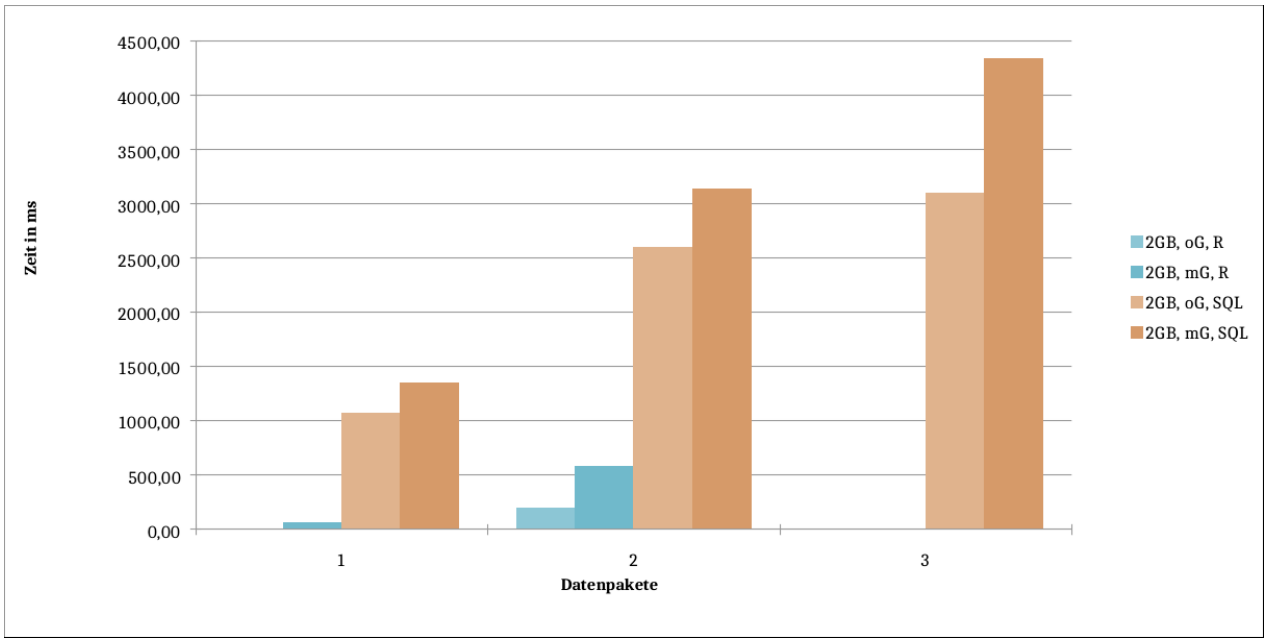
Grafik 5.1: Einlesen von Daten

Auffällig in Grafik 5.1 ist zunächst, dass `R` die Daten grundsätzlich schneller zur Verfügung stehen. Eine Ursache kann sein, dass `PgSQL` min. zwei Mal auf den Massenspeicher zugreifen muss: Zum einen, um den Datensatz einzulesen und zum anderen, diesen in die Tabelle zu schreiben. Besonders der Schreibvorgang dürfte sehr zeitintensiv sein. Diese Probleme hat `R`, zumindest wenn das Datenpaket komplett in den Arbeitsspeicher passt, nicht.
Betrachtet man dann zunächst die `SQL`-Werte (orangenfarbene Balken), erkennt man, dass die Hardwarekonfigurationen keinen Einfluss auf die Einlesezeiten besitzen. Ist man zunächst noch dazu geneigt, zu interpretieren, dass die Testprofile mit HDDs als Massenspeicher etwas länger benötigen, so wird dies beim großen Datenpaket, speziell bei 8GB, widerlegt. Denn die Einlesezeit beim HDD-Profil fällt leicht geringer aus als beim SSD-Profil.
Differenzierter muss man die Einlesezeiten bei `R` betrachten. Ins Auge fällt sofort, dass im großen Datenpaket zwei Messungen fehlen. Dieses ist dem Umstand geschuldet, dass `R` nicht in der Lage war, die Daten in einem angemessenem Zeitrahmen in das `Data.Frame`-Objekt einzulesen. Auch nach 12 Stunden lag noch kein Ergebnis vor (der Prozess selbst lief noch), weshalb dieser Testfall abgebrochen wurde. Wo im kleinen Datenpaket die Einlesezeiten noch

ausgeglichen sind, fällt beim mittleren und beim großen Datenpaket ein Gefälle auf (inkl. der längeren Einlesezeit von HDD-Massenspeichern). Hier spiegelt sich das Problem von `R` wider, dass der Massenspeicher hinzugezogen wird, wenn nicht genug Arbeitsspeicher zur Verfügung steht. Das sorgt für längere Einlesezeiten.
Beim Einlesen mit `R` wurde aber eine interessante Beobachtung gemacht: Nebenher wurde der Füllstand des Arbeitsspeichers und der Swappartition beobachtet (ohne diesen aufzuzeichnen). In dem Moment, als der Arbeitsspeicher voll war, wurde auf die Swappartition zugegriffen. In Intervallen, also wenn die Swappartition zu einem gewissen Grad gefüllt war, änderte sich kurzzeitig der Füllstand des Arbeitsspeichers signifikant (geschätzter Einbruch von rund 20%), um daraufhin wieder komplett gefüllt zu sein. Danach verringerte sich der benutzte Swappartitionsbereich. Aus diesem Verhalten könnte man interpretieren, dass `R` in vielleicht eine Reorganisation der Daten vornimmt, um (evtl.) die optimierten Daten in den Hauptspeicher zu verschieben. Nach 12 Stunden war dieses Verhalten nicht mehr zu beobachten.
Dies dürfte wohl der Grund sein, wieso das große Datenpakete mit den 2GB Arbeitsspeicher nicht eingelesen werden konnte, weil der reorganisierte Datenbestand ebenfalls nicht mehr in den Arbeitsspeicher passte. Allerdings ist dies reine Spekulation.

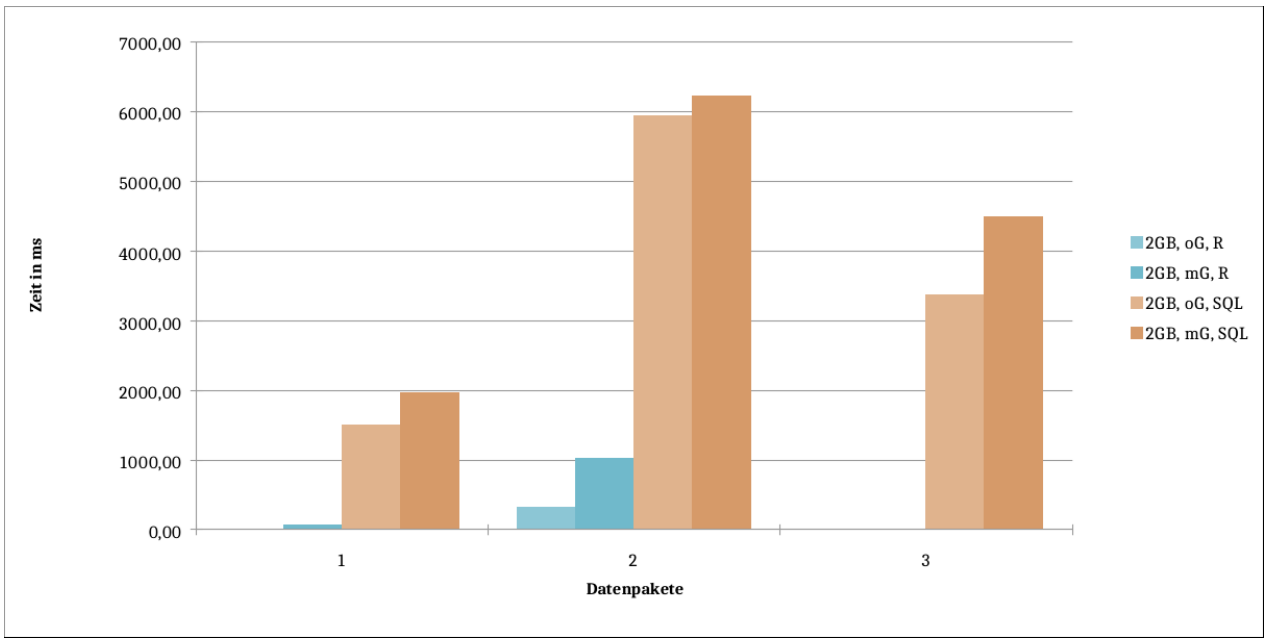
Testszenario #2: Statistische Berechnungen

Im zweiten Testszenario wurden die Berechnungszeiten für den Mittelwert, die Standardabweichung und die Varianz jeweils für `R` und `SQL` ermittelt. Diese gelten, wie in Testszenario #1, in direktem Vergleich zu betrachten.



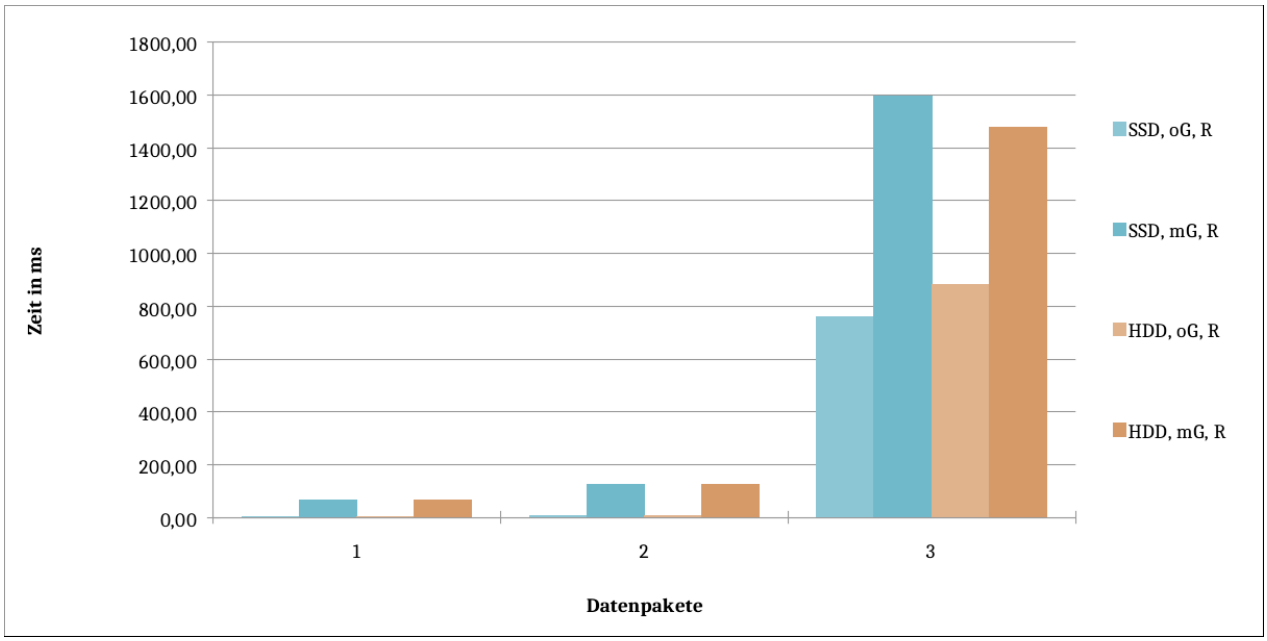
Grafik 5.2: Mittelwert bei 2GB Arbeitsspeicher und SSD

Wie in Testszenario #1, kann `SQL` bei der Berechnungsgeschwindigkeit nicht mithalten. Im kleinen Datenpaket kann man schlicht den Balken nicht mehr erkennen, da 4,31 ms zu gering sind. Dass die beiden `R`-Balken (und damit Berechnungszeiten) in Datenpaket 3 fehlen, liegt an dem Umstand, dass keine Daten zur Verfügung standen (siehe Interpretation von Testszenario #1). Unterschiede zwischen der Verwendung gruppierter und nicht gruppierter Daten sind gut zu erkennen. Berechnungen mit Gruppierung dauern in der Regel länger als ohne Gruppierung.



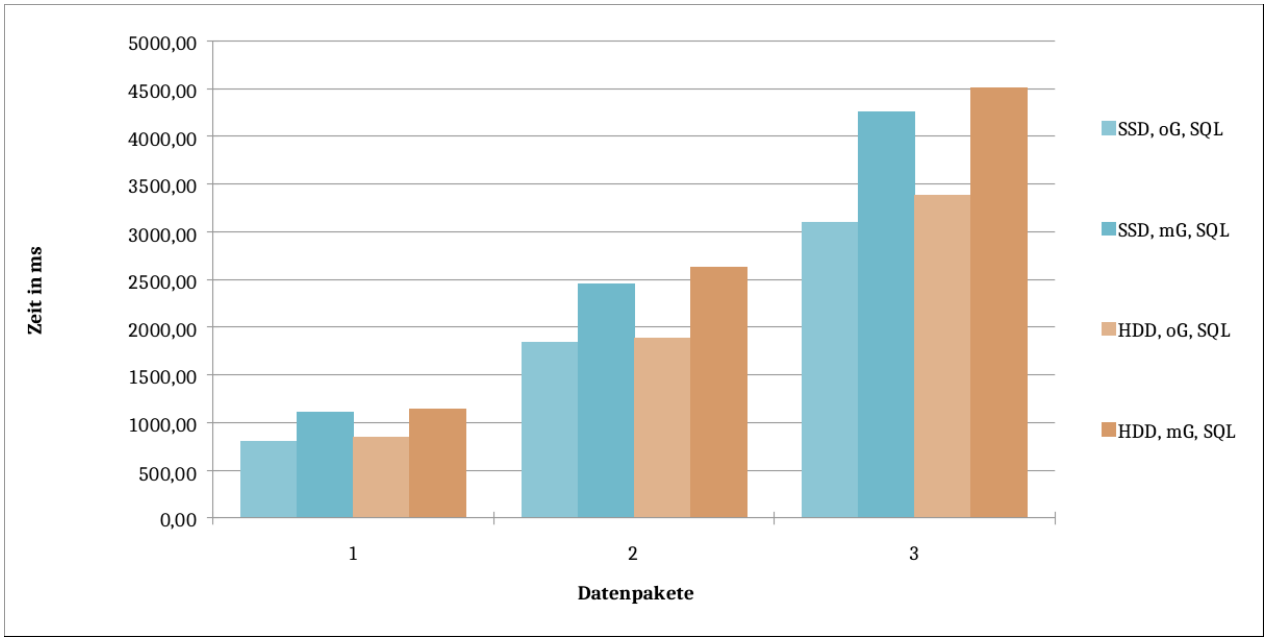
Grafik 5.3: Mittelwert bei 2GB Arbeitsspeicher und HDD

In Grafik 5.3 spiegelt sich ein ähnliches Bild wie in Grafik 5.2 wider. `R` ist schneller, 4,34ms sind zu wenig um angezeigt zu werden, beim großen Datenpaket fehlten Messwerte und die Gruppierung hat Auswirkungen auf die Berechnungszeit.



Grafik 5.4: Mittelwert bei 4GB in R

In Grafik 5.4 erkennt man wieder das Arbeitsspeicher-Problem von R, wenn die Daten nicht komplett in diesen reinpassen und die Swappartition benutzt werden musste. Die Berechnungszeiten steigern sich signifikant, wenn Daten ausgelagert werden müssen. Dass die Gruppierung Einfluss auf die Berechnungszeiten hat, wurde bereits in Grafik 5.3 festgestellt.

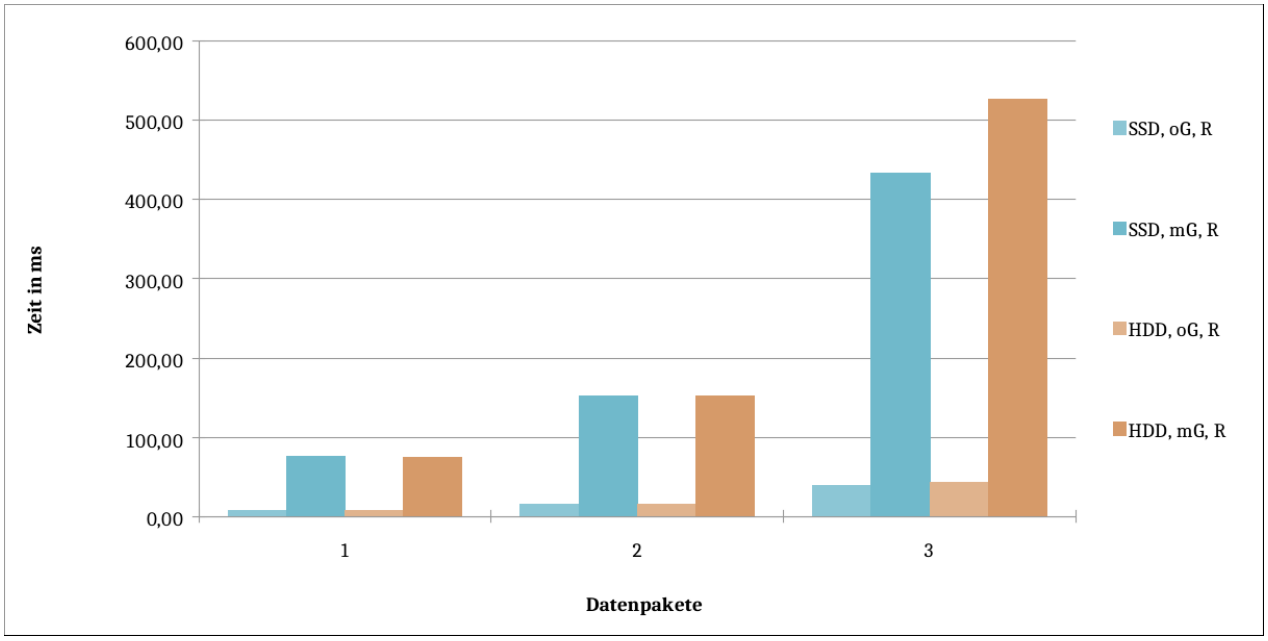


Grafik 5.5: Mittelwert bei 4GB mit PgSQL

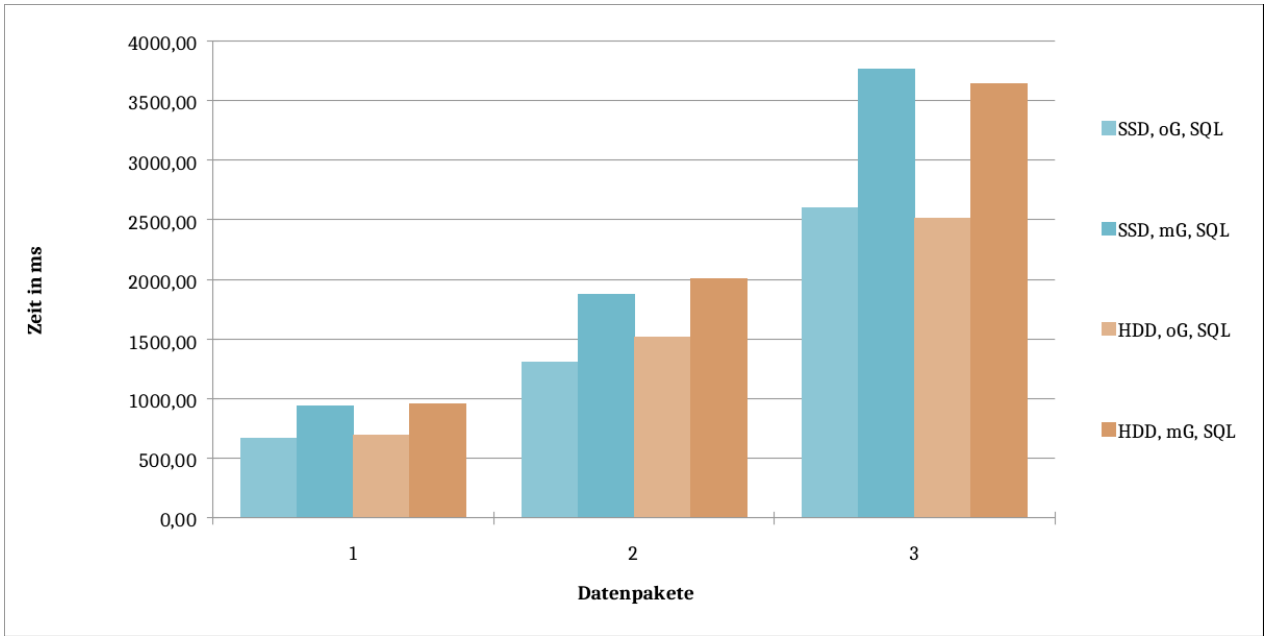
Wie mit Grafik 5.1 vergleichbar, zeigt sich `PgSQL` in Grafik 5.5 unbeeindruckt vom verwendeten Arbeitsspeicher und zeigt eine annähernd lineare Steigerung, wenn das Datenpaket in seiner Größe ebenfalls annähernd linear ansteigt. Einen Unterschied zwischen SSD und HDD kann man gerade beim großen Datenpaket sehr gut erkennen, bei dem die SSD leichte Vorteile

besitzt. Wie bei R in Grafik 5.4 hat die Gruppierung Auswirkungen auf die Berechnungszeit, weshalb die Berechnungen mit Gruppierung in der Regel länger dauern als ohne Gruppierung.

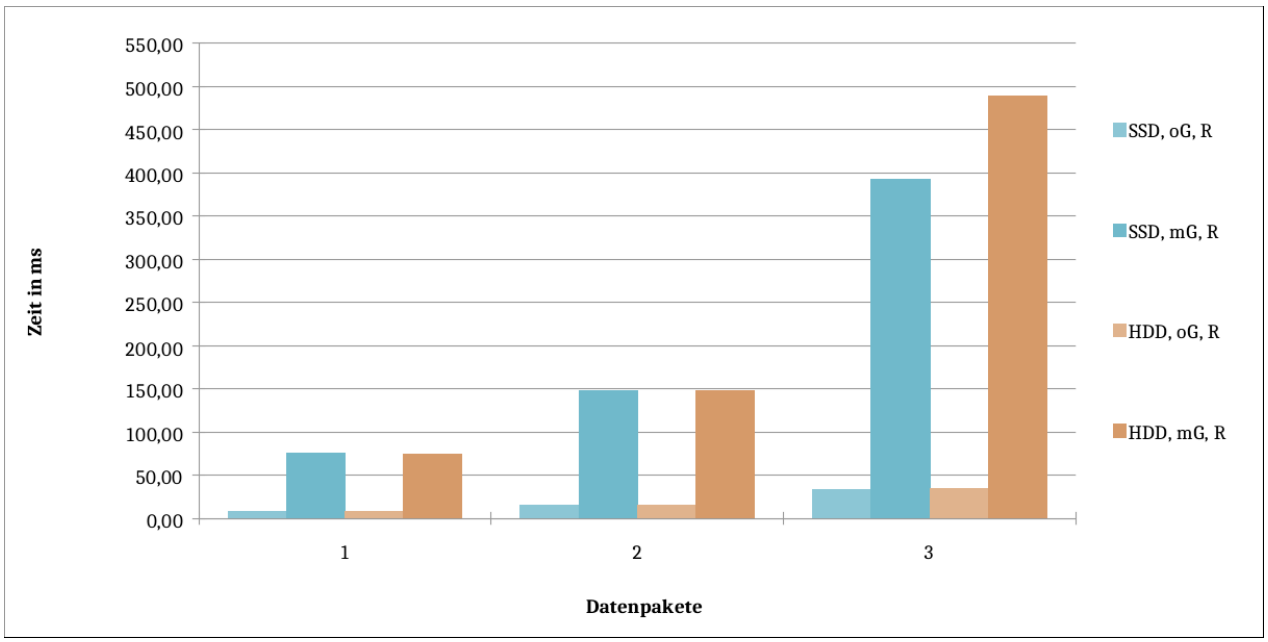
Betrachtet man nun die Tabellen für die Standardabweichung und die Varianz, zeichnet sich ungefähr das gleiche Bild wie bei den Mittelwerten ab, wie die Grafik 5.6, 5.7, 5.8 und 5.9 der Testprofile für 4GB belegen. Vergleicht man die Grafiken, lässt sich beim R-Mittelwert einen Unterschied zur Standardabweichung und Varianz feststellen. Es hat den Anschein, dass die Berechnung des Mittelwertes, wenn das Datenpaket nicht mehr komplett in den Arbeitsspeicher passt, signifikant länger dauert. Die Standardabweichung und Varianz reagieren auf diesen Umstand wesentlich unempfindlicher, sofern keine Gruppierung stattgefunden hat. Bei einer Gruppierung steigt die Berechnungszeit auch hier stark an, wenn auch nicht ganz so stark wie beim Mittelwert.



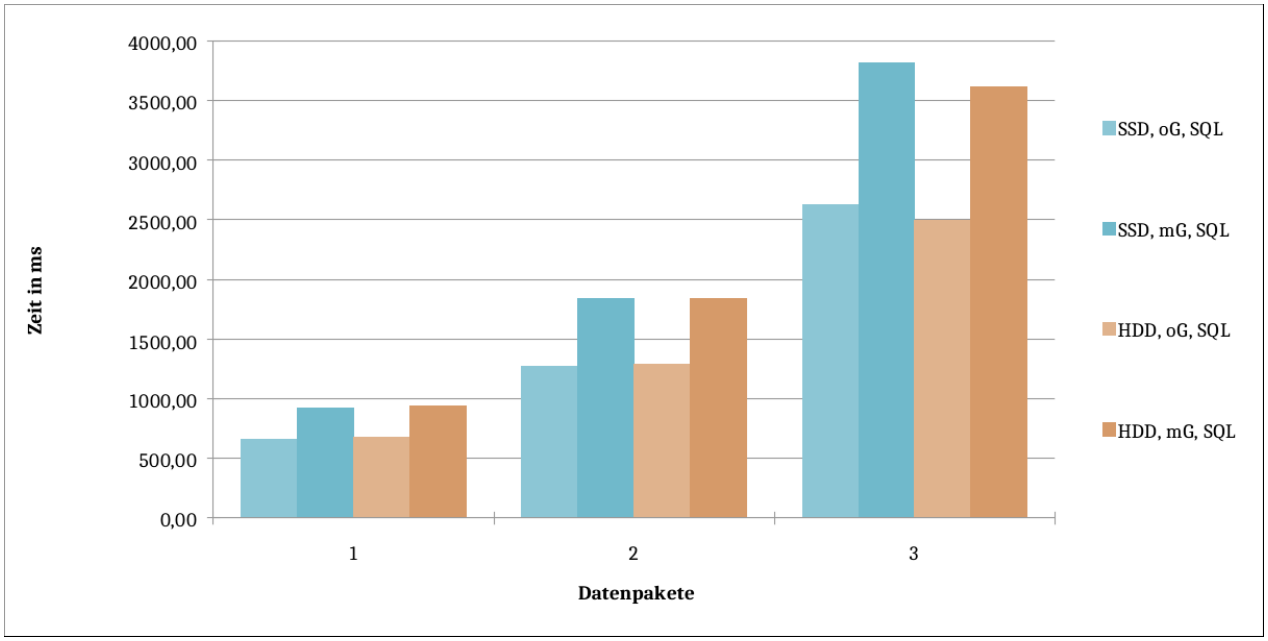
Grafik 5.6: Standardabweichung bei 4GB in R



Grafik 5.7: Standardabweichung bei 4GB in PgSQL

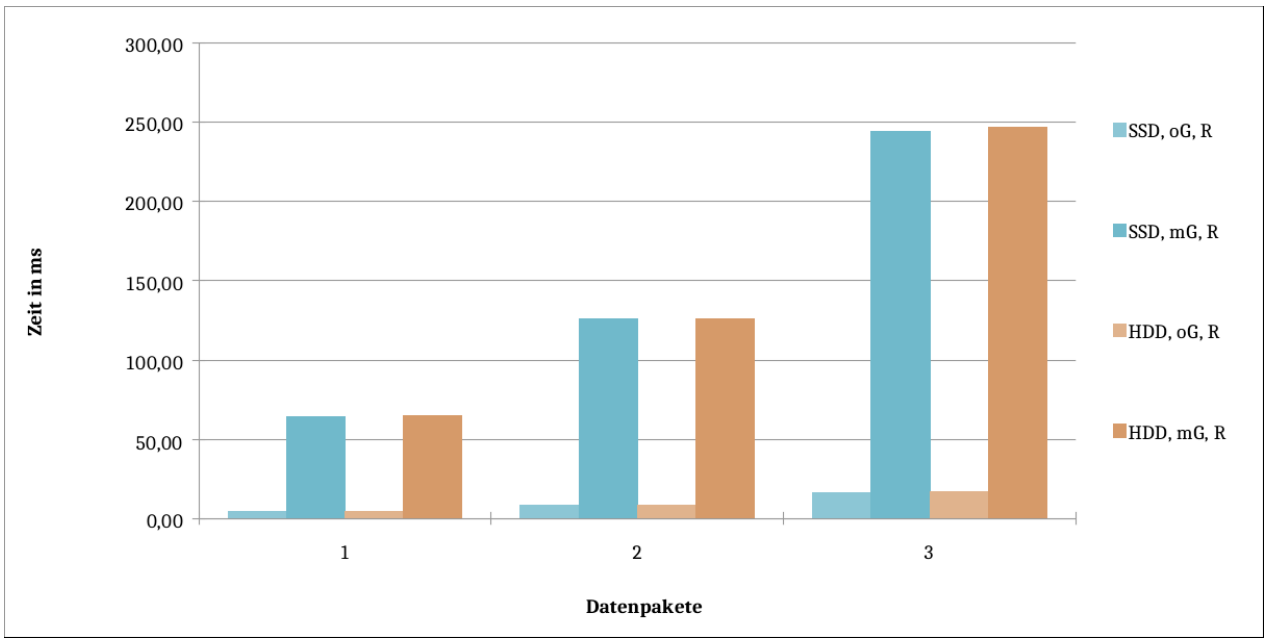


Grafik 5.8: Varianz bei 4GB in R

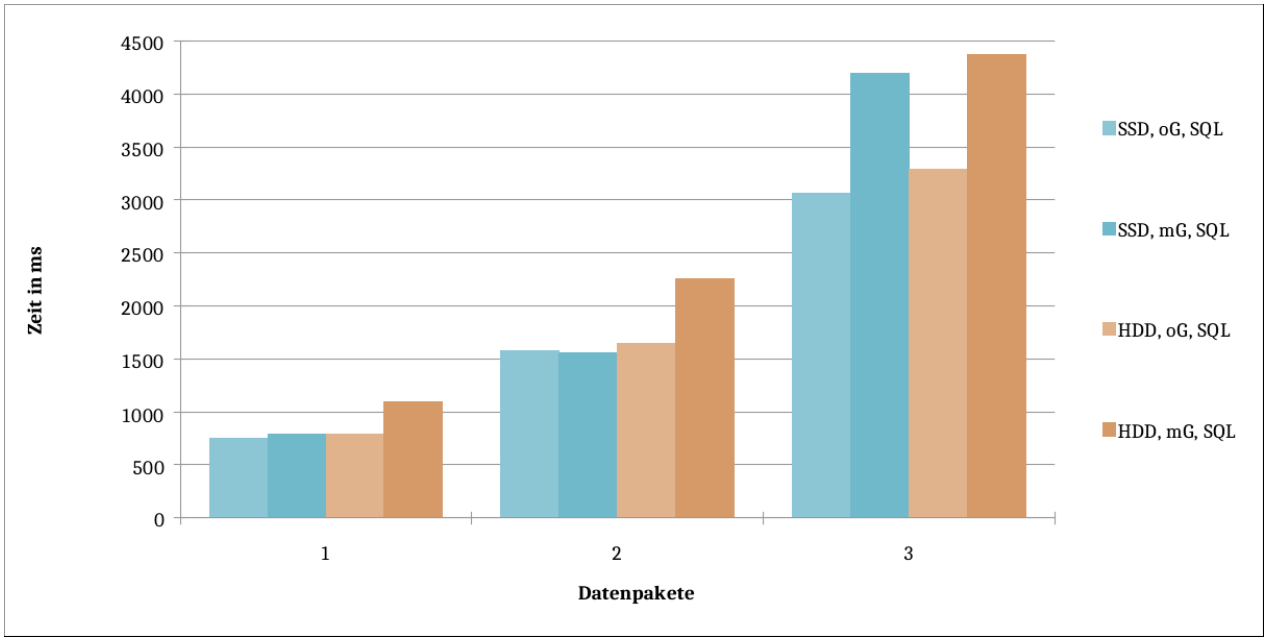


Grafik 5.9: Varianz bei 4GB in PgSQL

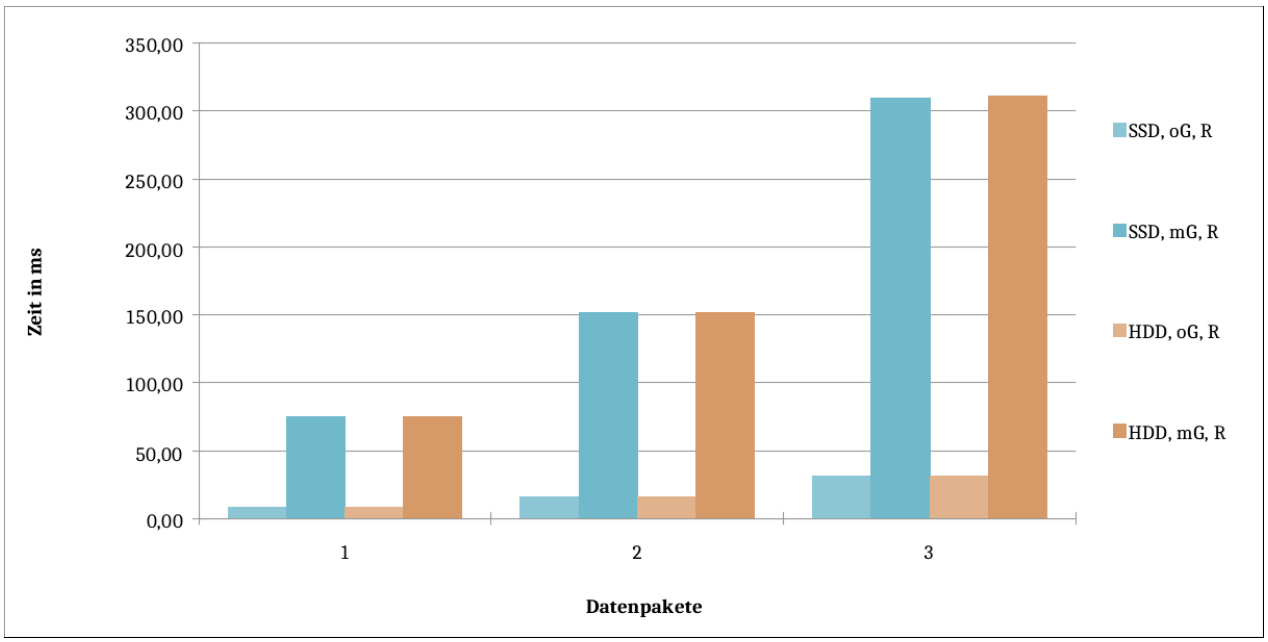
Schaut man sich nun die Testergebnisse für die Testprofile mit 8GB Arbeitsspeicher an (Grafik 5.10 bis 5.15), so ergeben sich bei `PgSQL`, vom Verhalten her, annähernd gleiche Ergebnisse. Auch hier hat die Gruppierung Einfluss auf die Berechnungszeit. Interessant ist jedoch, dass der Einsatz von herkömmlichen HDD-Festplatten kaum Einfluss auf die Berechnungszeit haben können. Lediglich bei der Standardabweichung und der Varianz im Bezug zum großen Datenpaket ergibt sich hier ein gemischtes Bild.
`R` zeigt auch hier, dass sich die Gruppierung erheblich auf die Berechnungszeit auswirkt. Wie zu erwarten, ist beim großen Datenpaket der Mittelwert allerdings auf einem linearen Level zu den beiden anderen Datenpaketen geschrumpft.



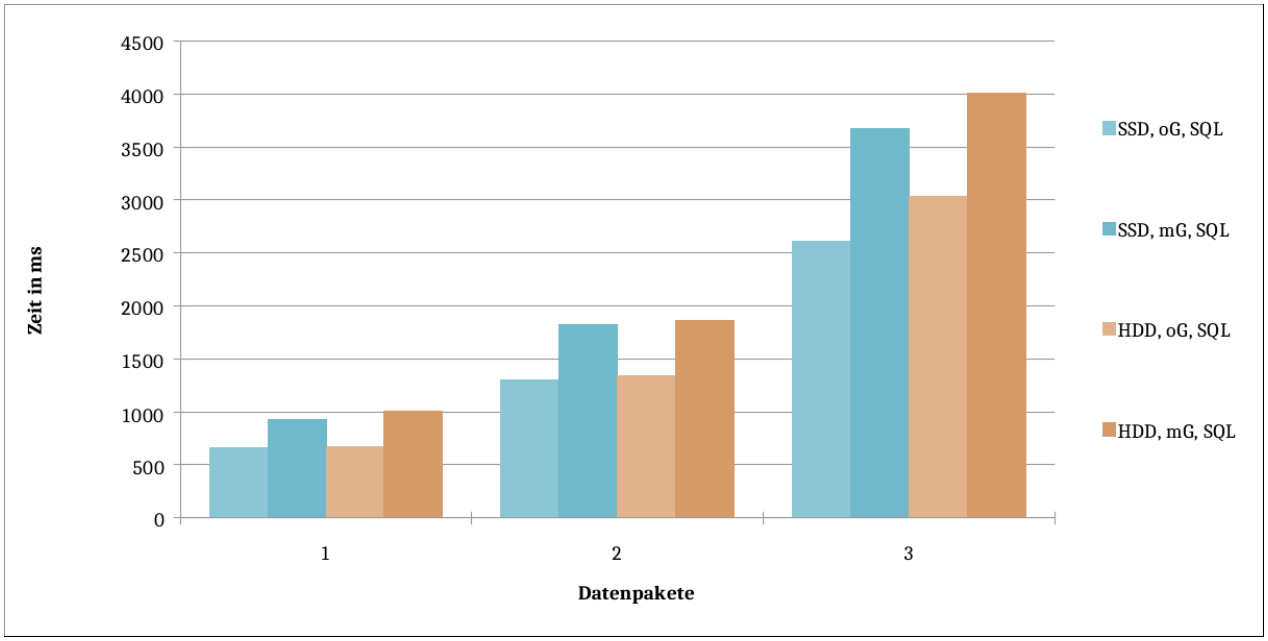
Grafik 5.10: Mittelwert bei 8GB in R



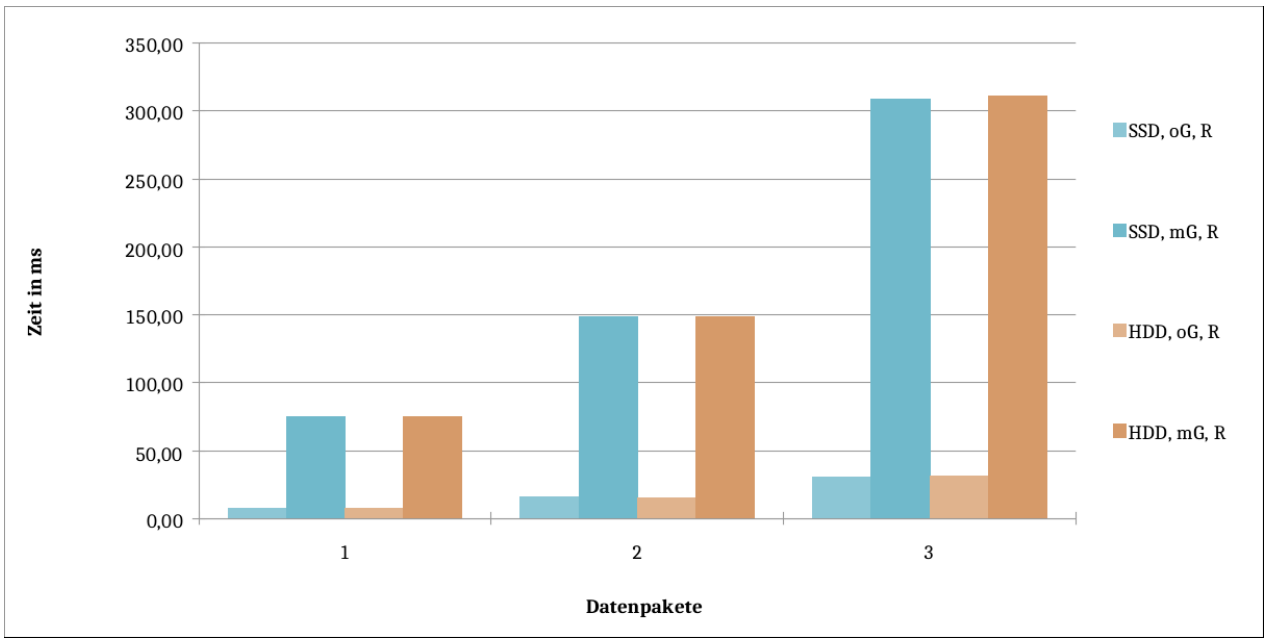
Grafik 5.11: Mittelwert bei 8GB in PgSQL



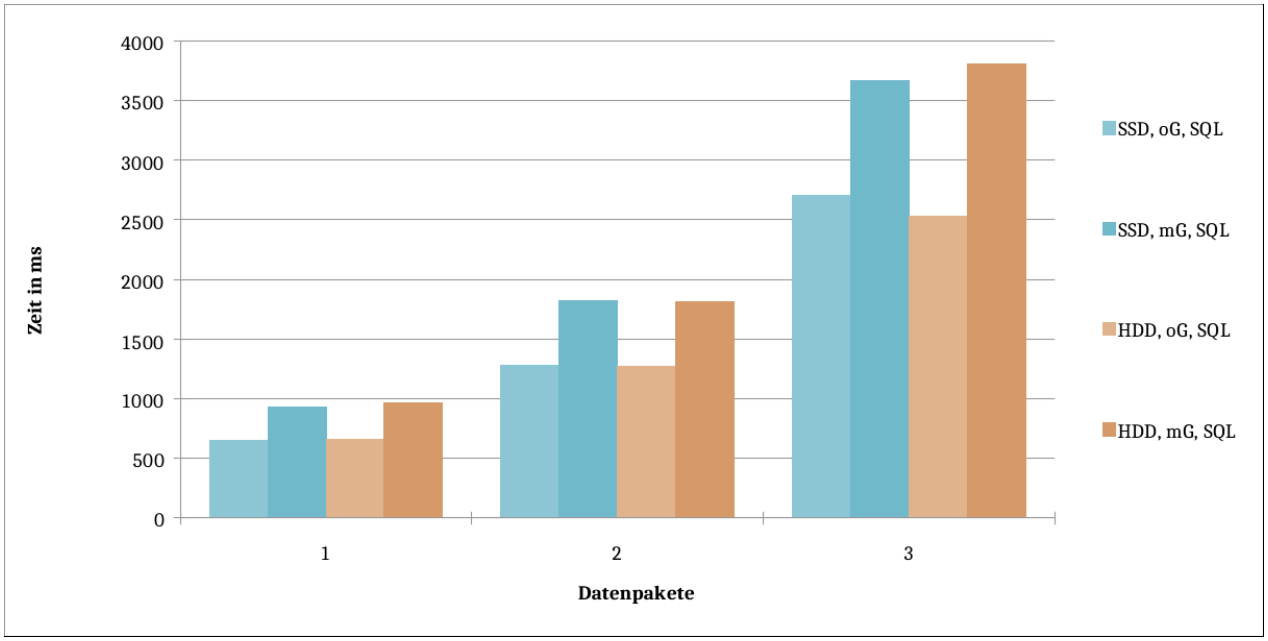
Grafik 5.12: Standardabweichung bei 8GB in R



Grafik 5.13: Standardabweichung bei 8GB in PgSQL



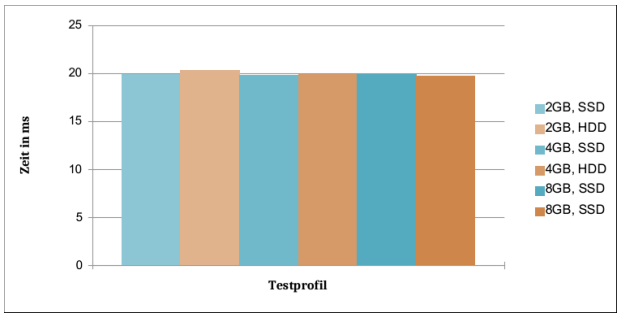
Grafik 5.14: Varianz bei 8GB in R



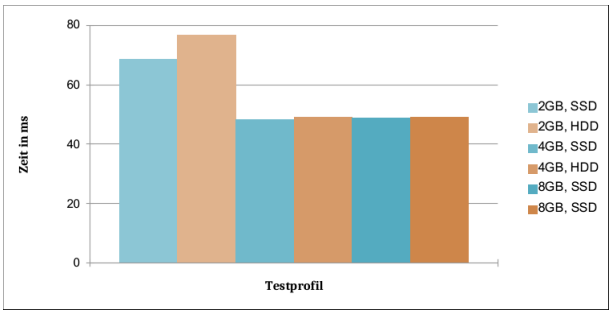
Grafik 5.15: Varianz bei 8GB in PgSQL

Testszenario #3: PL/R und fread()

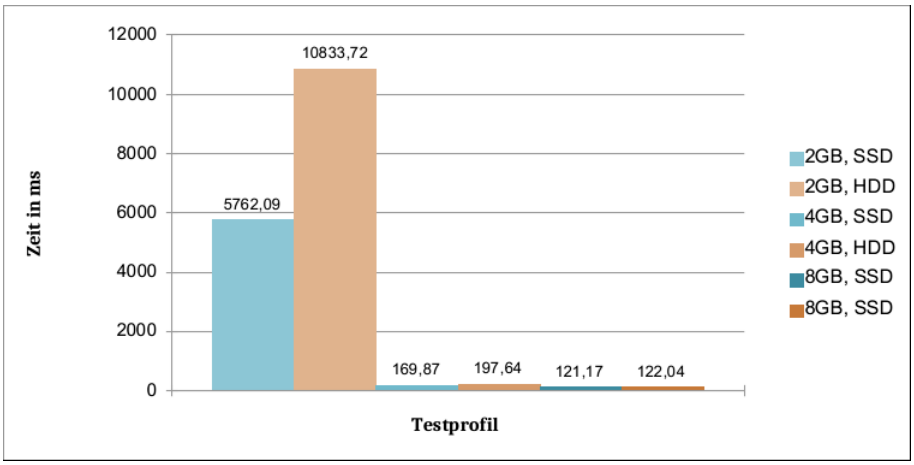
Die Grundwerte sind gegeben, nun wird verglichen, inwiefern sich die gemessenen Zeiten bei der Kopplung von `R` und `SQL` zu den Grundwerten aus Testszenario #1 und #2 unterscheiden. Doch zunächst sollen die Messergebnisse für sich betrachtet werden.



Grafik 5.16: kleines Datenpaket



Grafik 5.17: mittleres Datenpaket



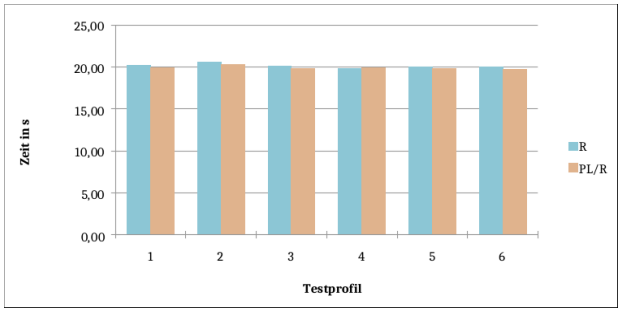
Grafik 5.18: großes Datenpaket

In Grafik 5.16 sieht man die Einlesezeiten des kleinen Datenpaketes. Da dieses in den Arbeitsspeicher aller Testprofile passte, sind die Zeiten in etwa identisch.

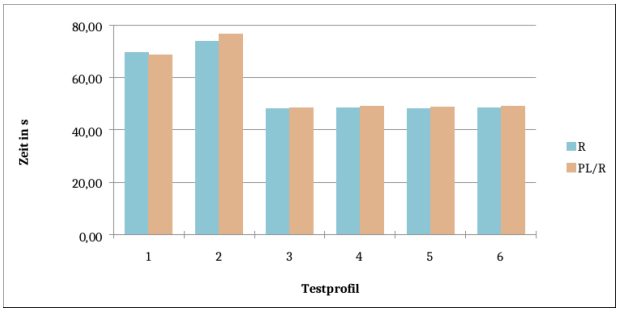
Grafik 5.17 wird schon interessanter. Die Testprofile mit 4GB und 8GB ergeben wieder ungefähr die gleichen Einlesezeiten, nur die für 2GB schlagen nach oben. Das entspricht den Erwartungen, da das mittlere Datenpaket nicht in die 2GB Arbeitsspeicher passen und damit die Swappartition benutzt werden musste.

Die interessanteste Grafik ist 5.18, da diese eine unerwartete Überraschung enthält. Schaut man sich allerdings zuerst die 4GB- und 8GB-Profile an, so kann man hier gut erkennen, dass das große Datenpaket nicht in die 4GB passte und die Zeiten deshalb größer sind als die vom 8GB-Testprofil. Die eigentliche Überraschung ist aber, dass die 2GB-Testprofile in der Lage waren, das große Datenpaket in ein `Data.Frame`-Objekt unterzubringen. Scheiterte dies noch in Testszenario #1, so ergeben sich hier immerhin Einlesezeiten von rund 5762 Sekunden (~96 Minuten) für die SSD-Konfiguration und 10833 Sekunden (~181 Minuten) für die HDD-Konfiguration. Dies wirft einige Fragen auf. Wie ist es `PL/R` gelungen, das große Datenpaket gelungen, woran die reine `R`-Umgebung scheiterte? Hier könnte evtl. die Vermutung aufkommen, basierend auf der Spekulation in Testszenario #1, dass evtl. die Reorganisation der Daten im Arbeitsspeicher, bzw. im Swapbereich anders vorgenommen wird. Aber auch dies ist wieder reine Spekulation. Nichts desto trotz dürfte es interessant werden, wie die Berech-

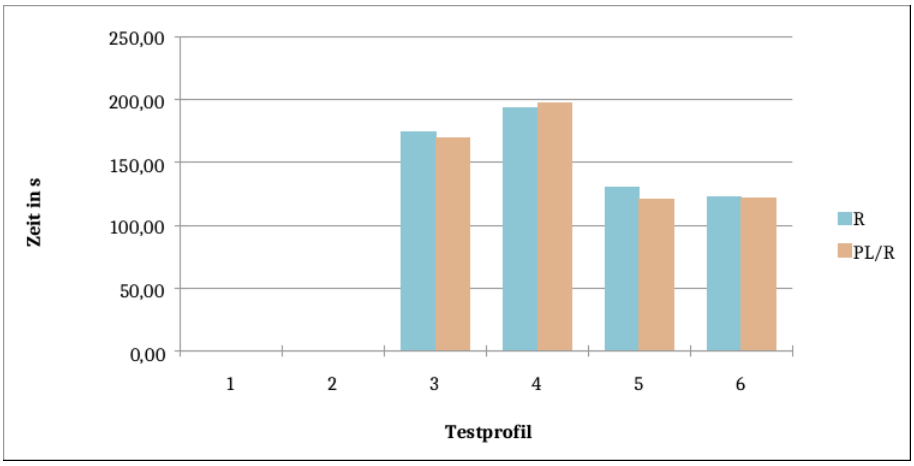
nungszeiten in Testszenario #5 abgeschnitten haben.



Grafik 5.19: kleines Datenpaket



Grafik 5.20: mittleres Datenpaket

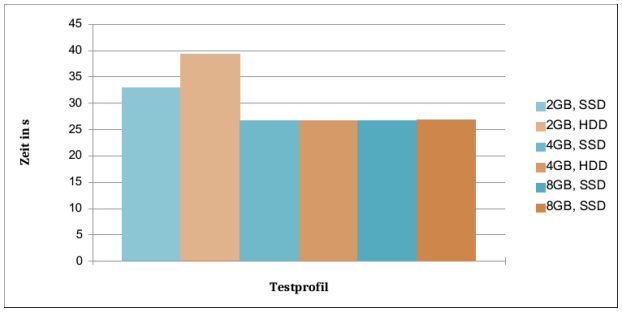


Grafik 5.21: großes Datenpaket

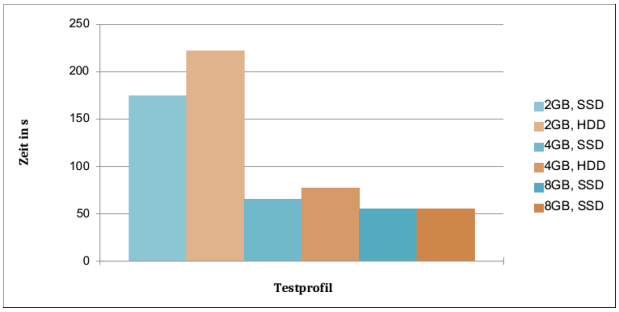
Vergleicht man nun die Einlesezeiten, wobei in Grafik 5.21 die beiden ersten Testprofile wegen der besseren Übersicht ausgeblendet wurden (die reine R-Umgebung konnte sowieso keine Vergleichswerte liefern) mit denen aus Testszenario #1, kann man, abgesehen von Messtoleranzen, keinen bemerkenswerten Unterschied feststellen. Die Einlesezeiten sind nahezu identisch.

Testszenario #4: PL/R und pg.spi.exec()

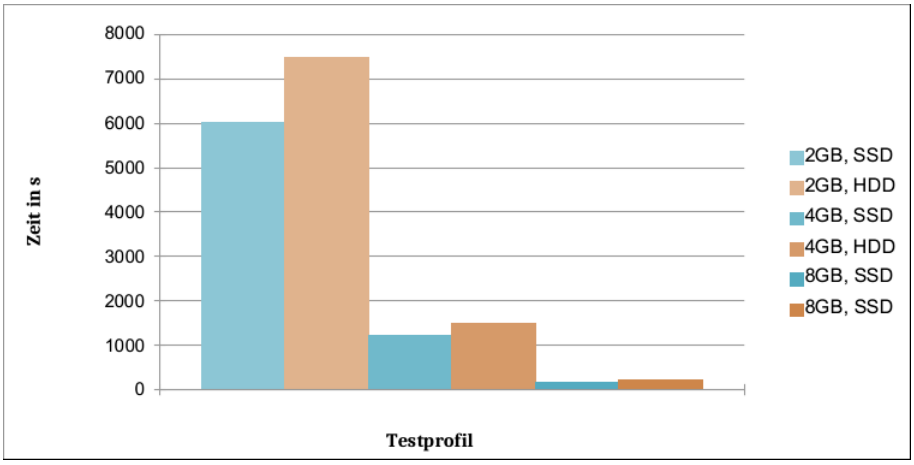
Die andere Variante, um an die Daten zu kommen, ist die Kapazitäten der Datenbank auszunutzen und sich die Datensätze per `SQL`-Befehl in ein Data.Frame einzulesen. Da sich die Einlesezeiten von Testszenario #1 und #3 nahezu gleichen, wurde Testszenario #3 als Vergleich herangezogen (vor allem wegen dem Vergleichswert für die 2GB-Testprofile im Bezug zum großen Datenpaket). Doch zunächst sollen die Messwerte wieder für sich selbst betrachtet werden.



Grafik 5.22: kleines Datenpaket



Grafik 5.23: mittleres Datenpaket

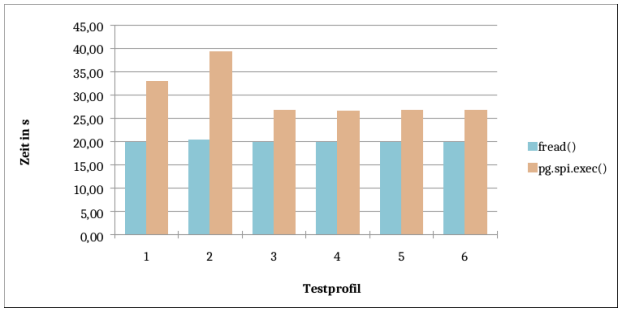


Grafik 5.24: großes Datenpaket

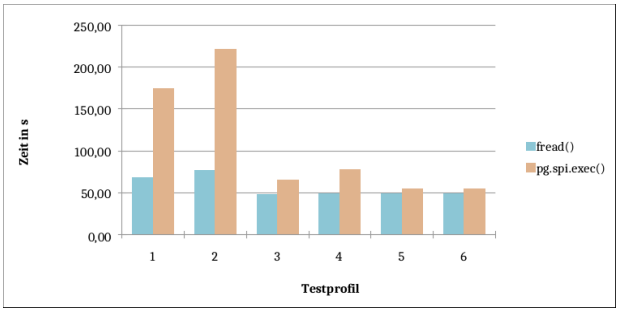
Die Einlesezeiten für das kleine Datenpaket wurde in Grafik 5.22 gemessen. Auffällig ist, dass die 2GB-Testprofile eine erhöhte Einlesezeit besitzen, woraus man schließen kann, dass der Arbeitsspeicher nicht gereicht hat und die Swappartition benutzt werden musste. Welche großen Auswirkungen das haben kann, zeigt Grafik 5.23 (mittleres Datenpaket). Die 2GB-Testprofile haben eine stark erhöhte Einlesezeit, wohingegen die 4GB-Profile eine etwas erhöhte Einlesezeit besitzen. Auch dies spricht wieder dafür, dass das Datenpaket nicht komplett in den Arbeitsspeicher gepasst hat. Die Auswirkungen lassen sind Grafik 5.24 am Besten erkennen, wo das große Datenpaket eingelesen werden musste. Schön zu sehen ist auch die Tatsache, dass die HDD-Profile länger benötigten als die SSD-Profile. Und auch hier zeigt `PL/R`, wie bereits in Testszenario #3, dass große Datenmengen bei geringem Arbeitsspeicher kein Problem sind. Die 2GB-Testprofile waren in der Lage, alle Daten in das `Data.Frame`-Objekt zu füllen.

Vergleicht man nun diese Einlesezeiten mit denen aus Testszenario #3, erkennt man eindeutig (mit einer Ausnahme), dass `fread()` die Daten stets schneller eingelesen hat. Die Ausnahme beim großen Datenpaket und TP#2 kann so nicht erklärt werden, vor allem weil die HDD-Konfiguration extrem aus dem Rahmen schlägt. Ursache könnte möglicherweise ein

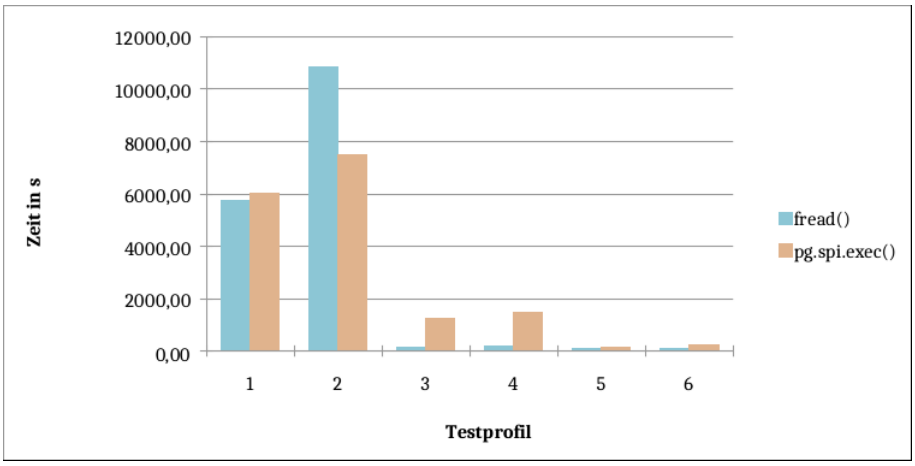
Messfehler sein, der evtl. nochmal genauer hätte untersucht werden müssen.



Grafik 5.25: kleines Datenpaket



Grafik 5.26: mittleres Datenpaket



Grafik 5.27: großes Datenpaket

Betrachtet man nun die Anstiegszeiten, also wie der prozentuale Unterschied aussieht, kommen dabei folgende Ergebnisse zustande:

	TP#1	TP#2	TP#3	TP#4	TP#5	TP#6
fread()	19,91	20,34	19,80	19,91	19,86	19,76
pg.spi.exec()	32,98	39,37	26,69	26,62	26,65	26,77
%	65,65	93,56	34,80	33,70	34,19	35,48

Tabelle 5.1: Anstiege, kleines Datenpaket

	TP#1	TP#2	TP#3	TP#4	TP#5	TP#6
fread()	68,48	76,67	48,29	49,00	48,83	48,95
pg.spi.exec()	174,44	221,60	65,22	77,59	54,82	54,95
%	154,73	189,03	35,06	58,35	12,27	12,26

Tabelle 5.2: Anstiege, mittleres Datenpaket

	TP#1	TP#2	TP#3	TP#4	TP#5	TP#6
fread()	5762,09	10833,72	169,87	197,64	121,17	122,04
pg.spi.exec()	6022,76	7487,57	1229,50	1501,02	166,21	229,65
%	4,52	-30,89	623,79	659,47	37,17	88,18

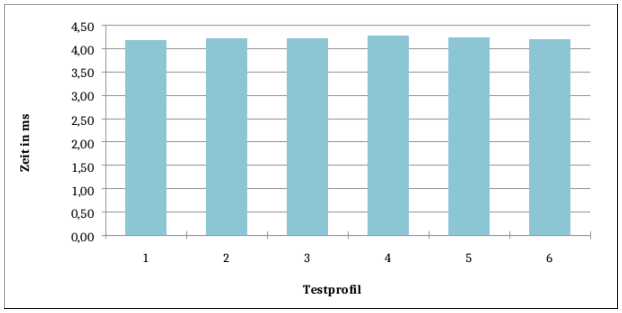
Tabelle 5.3: Anstiege, großes Datenpaket

Diese Werte zu interpretieren erfordert etwas Fingerspitzengefühl. Was gesagt werden kann, ist, dass wenn das Datenpaket sicher in den Arbeitsspeicher passt, der Anstieg am geringsten ist. Man betrachte dazu in Tabelle 5.1 TP#3, TP#4, TP#5 und TP#6, bzw. in Tabelle 5.2 TP#5 und TP#6. Sollten die Datenpakete beim besten Willen nicht in den Arbeitsspeicher passen, nehmen sich beide Einleseverfahren so gut wie nichts. Dies gilt nur unter der Annahme, wenn

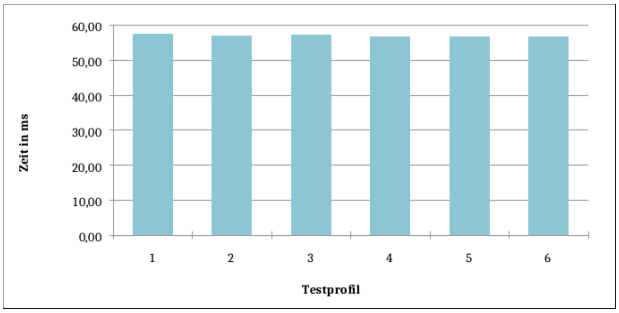
in Tabelle 5.2 mit TP#2 bei `fread()` ein Messfehler stattgefunden hat. Wenn nicht, kann diese Aussage entweder nicht gehalten werden oder man muss zwischen SSD und HDD unterscheiden. Deshalb wäre es angebrachter, dass das Verhalten in späteren Arbeiten genauer untersucht werden sollte, um haltbare Aussagen darüber treffen zu können. So bleibt es dabei, dass `fread()` in der Regel schneller ist als `pg.spi.exec()`.

Testszenario #5: PL/R und statistische Berechnungen

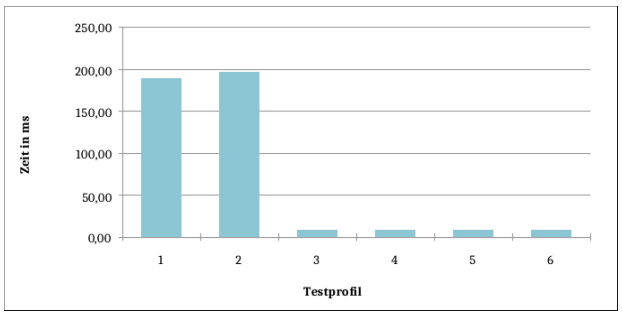
Testszenario #3 hat gezeigt, dass die Integration von `R` in `PgSQL`, zumindest was bisher die Einlesezeiten betrifft, keine Nachteile mit sich bringt. Im Gegenteil, in `PL/R` war es möglich, Datensätze einzulesen, die in der reinen `R`-Umgebung keine Ergebnisse hervorbrachten (siehe Kapitel 5, Testszenario #1). Umso interessanter dürfte der Vergleich mit Testszenario #2 werden, welcher in diesem Teil der Interpretation genauer vorgenommen wird. Doch zunächst sollen wieder die Testergebnisse für sich betrachtet werden.



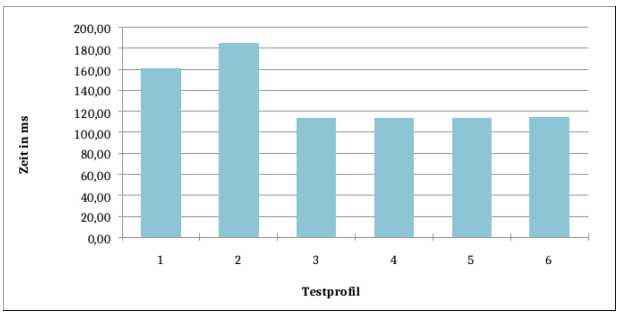
Grafik 5.28: Mittelwert, low, o.G.



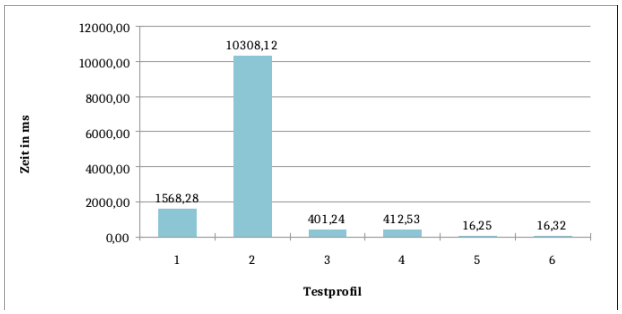
Grafik 5.29: Mittelwert, low, m.G.



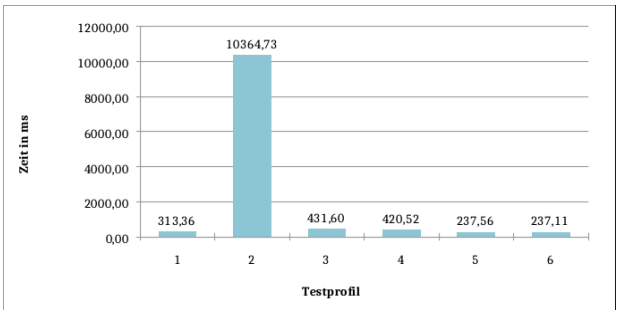
Grafik 5.30: Mittelwert, med, o.G.



Grafik 5.31: Mittelwert, med, m.G.

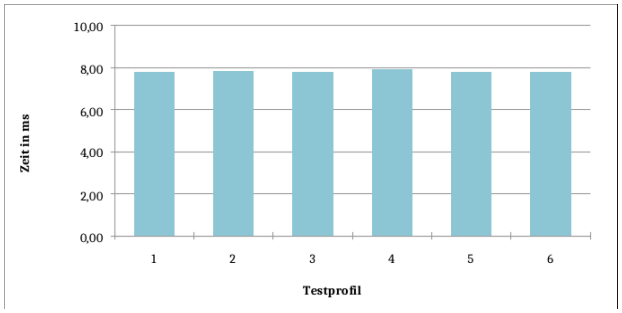


Grafik 5.32: Mittelwert, high, o.G.

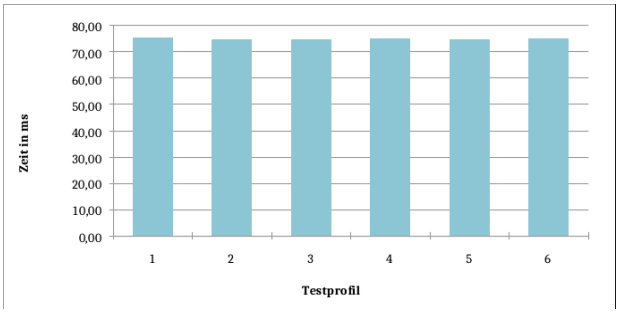


Grafik 5.33: Mittelwert, high, m.G.

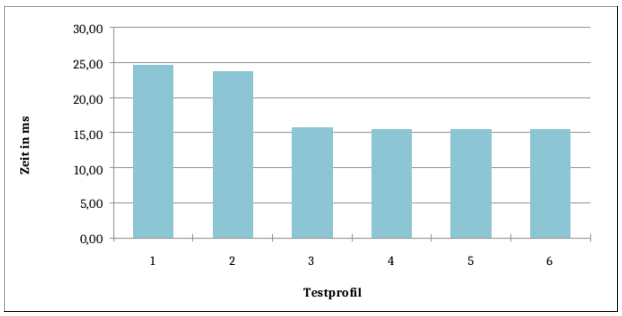
Die Mittelwerte offenbaren keine Überraschungen. Solange die Daten in den Arbeitsspeicher passen, sind die Berechnungszeiten gleichmäßig (Grafik 5.28 und 5.29). In Grafik 5.30 und 5.31 ist wieder deutlich zu erkennen, dass das mittlere Datenpaket nicht in den Arbeitsspeicher passt. Dementsprechend schießen die Berechnungszeiten auch nach oben, was in Grafik 5.32. und 5.33 wesentlich deutlicher zu sehen ist.



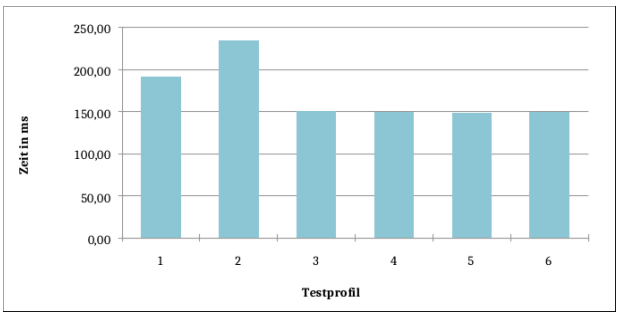
Grafik 5.34: Standardabw., low, o.G.



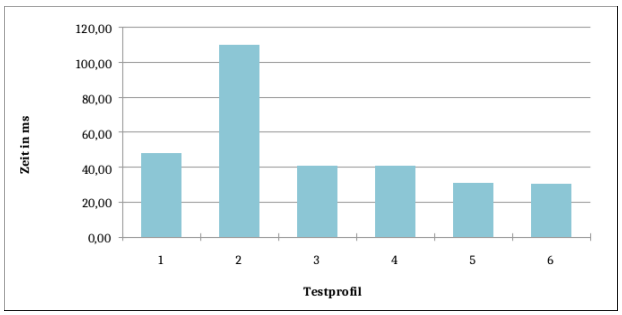
Grafik 5.35: Standardabw., low, m.G.



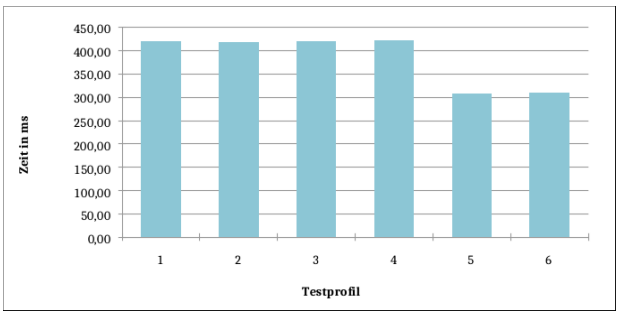
Grafik 5.36: Standardabw., med, o.G.



Grafik 5.37: Standardabw., med, m.G.

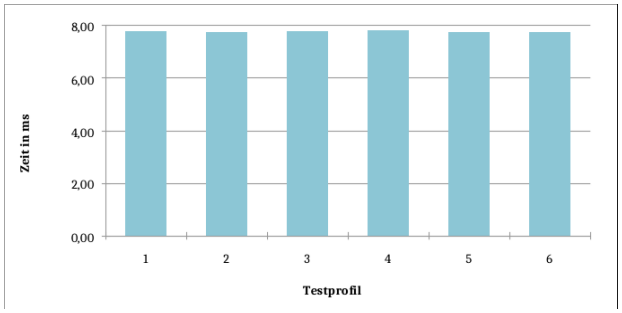


Grafik 5.38: Standardabw., high, o.G.

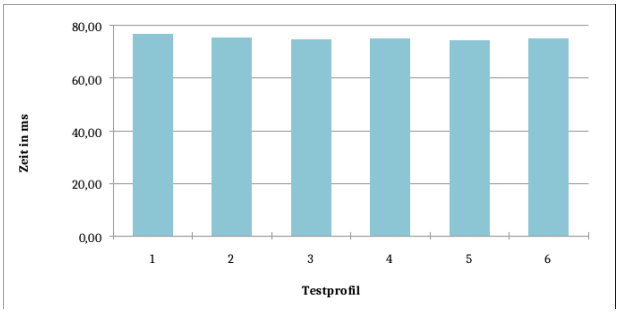


Grafik 5.39: Standardabw., high, m.G.

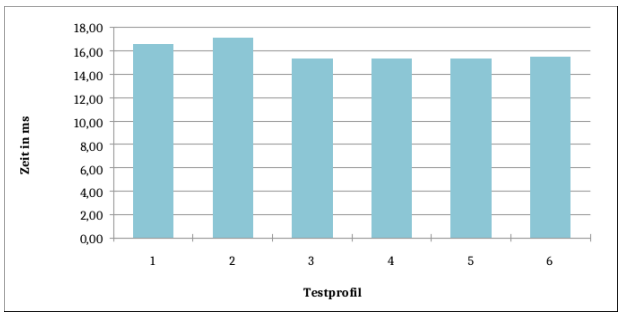
Prinzipiell kann bei der Standardabweichung genau das gleiche geschrieben werden wie zu den Mittelwerten. Die Ausnahme stellt Grafik 5.38 dar, wo die Auswirkungen des sehr knappen Arbeitsspeichers nicht so drastisch ist wie beim Mittelwert (Grafik 5.32).



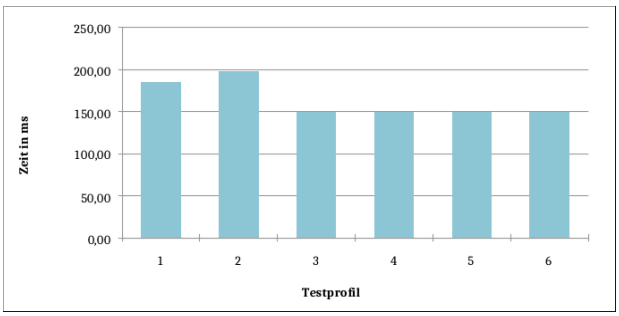
Grafik 5.40: Varianz, low, o.G.



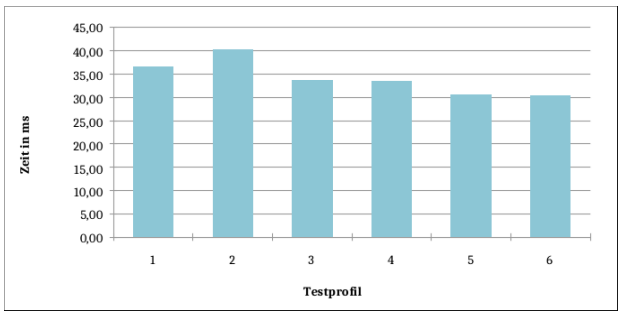
Grafik 5.41: Varianz, low, m.G.



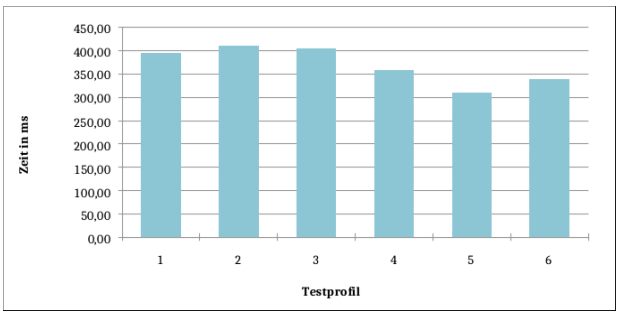
Grafik 5.42: Varianz, med, o.G.



Grafik 5.43: Varianz, med, m.G.



Grafik 5.44: Varianz, high, o.G.

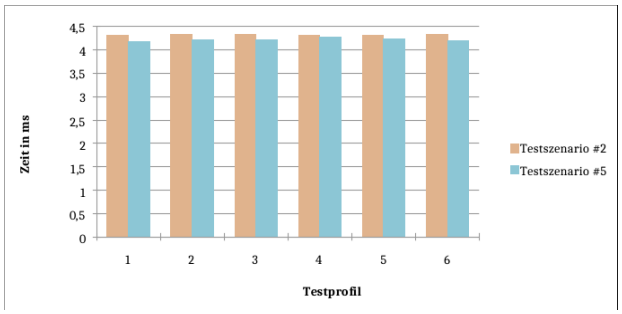


Grafik 5.45: Varianz, high, m.G.

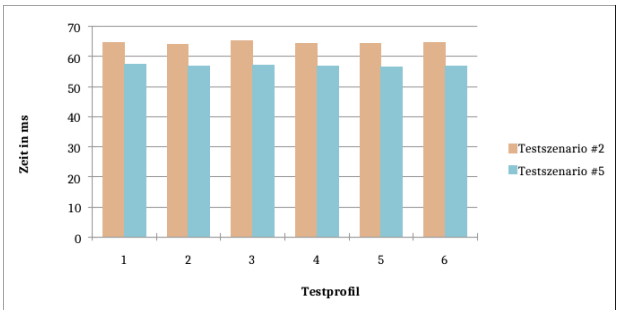
Und auch die Varianz birgt keine weiteren Überraschungen. Interessant ist jedoch, dass die Auswirkungen des knappen Arbeitsspeichers noch besser abgefedert werden als in der Standabweichung (vgl. Grafik 5.44 mit Grafik 5.38 und 5.32).

Zusammenfassend kann also gesagt, dass bis zu diesem Punkt die Ergebnisse, basierend auf den Erfahrungen aus Testszenario #2, weitestgehend den Erwartungen entsprechen. Doch wie schlagen sich diese Werte mit denen aus Testszenario #2? Dieser Frage wird nun ausgiebig nachgegangen.

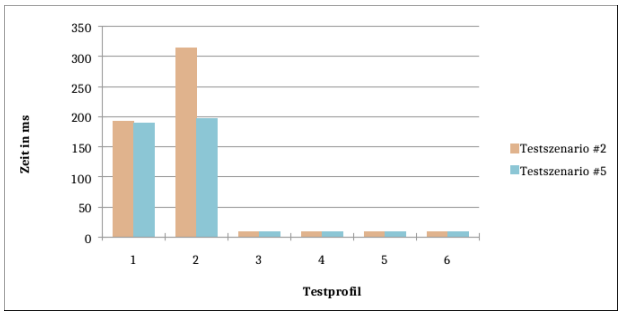
Verglichen werden sollen zunächst die Mittelwerte aus Testszenario #2 mit den oben ermittelten Berechnungszeiten.



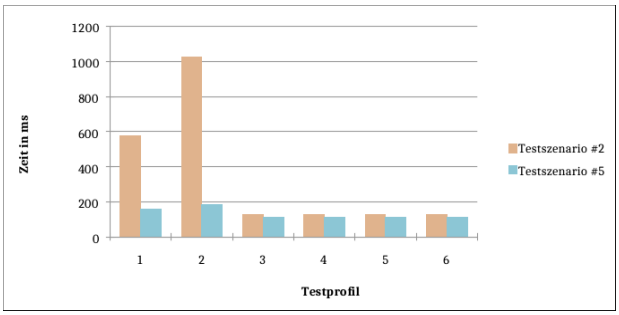
Grafik 5.46: Mittelwert, low, o.G.



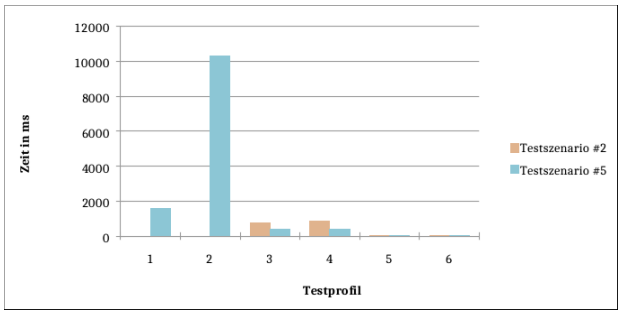
Grafik 5.47: Mittelwert, low, m.G.



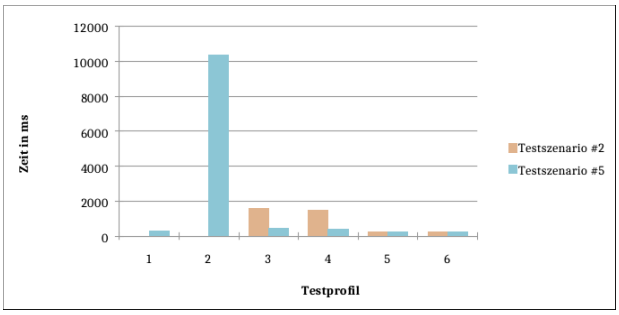
Grafik 5.48: Mittelwert, med, o.G.



Grafik 5.49: Mittelwert, med, m.G.

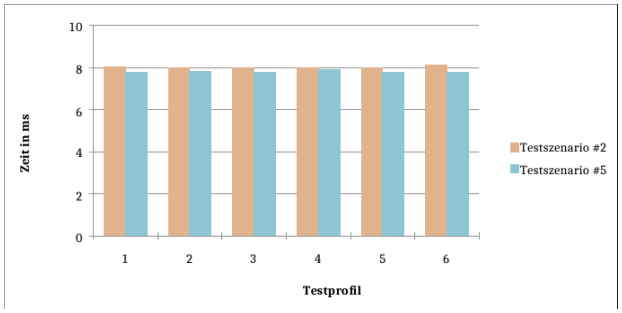


Grafik 5.50: Mittelwert, high, o.G.

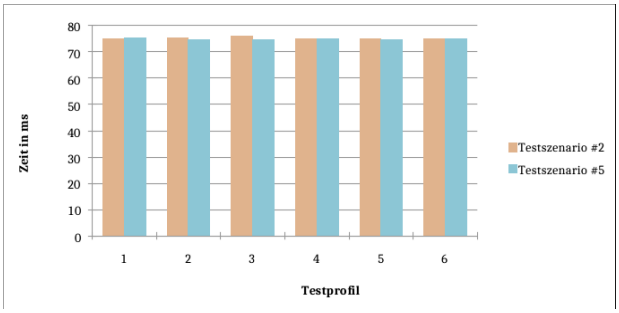


Grafik 5.51: Mittelwert, high, m.G.

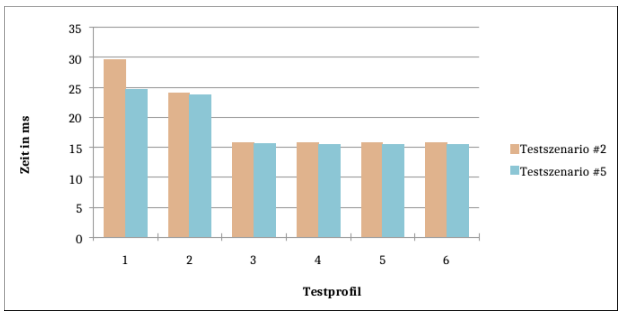
Wie man anhand der Grafiken 5.46 bis 5.51 erkennen kann, sind die Berechnungen innerhalb von `PL/R` ausnahmslos schneller erledigt. Zudem sollte man nicht vergessen, dass `PL/R` für 2GB-Testprofile mit großen Datenpaket immerhin ein Ergebnis liefert.



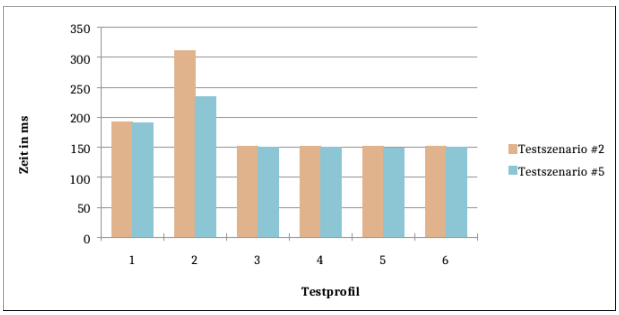
Grafik 5.52: Standardabw., low, o.G.



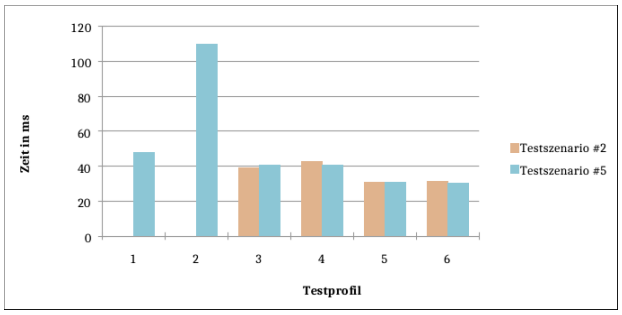
Grafik 5.53: Standardabw., low, m.G.



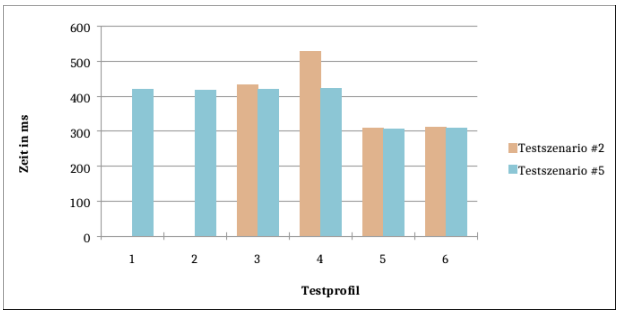
Grafik 5.54: Standardabw., med, o.G.



Grafik 5.55: Standardabw., med, m.G.

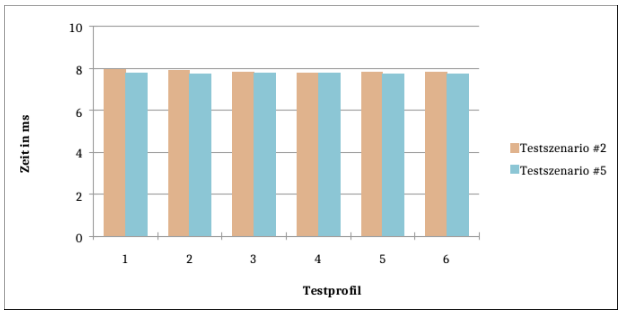


Grafik 5.56: Standardabw., high, o.G.

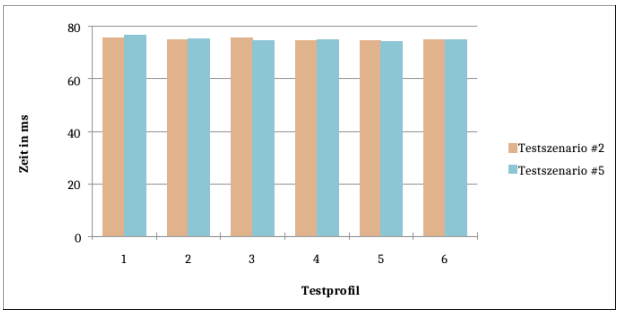


Grafik 5.57: Standardabw., high, m.G.

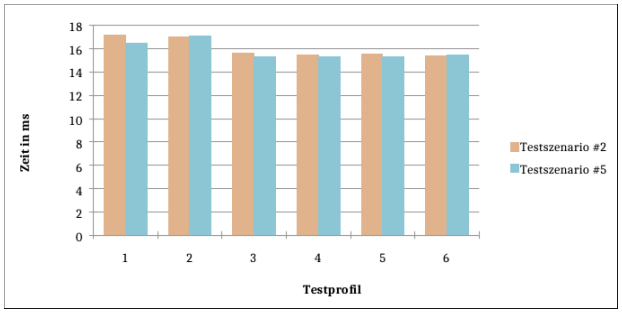
Bei der Berechnung der Standardabweichung ergibt sich zwar ein nicht mehr eindeutiges Bild, dennoch überwiegt die Anzahl der kürzeren Berechnungszeiten zugunsten PL/R.



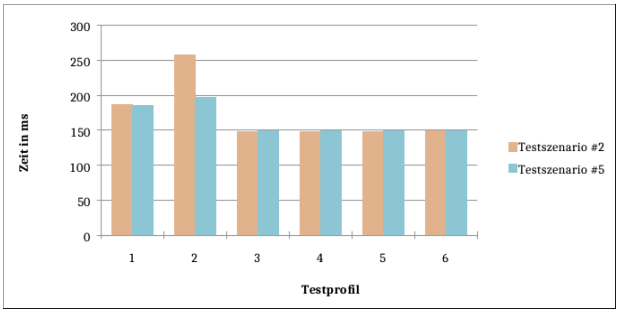
Grafik 5.58: Varianz, low, o.G.



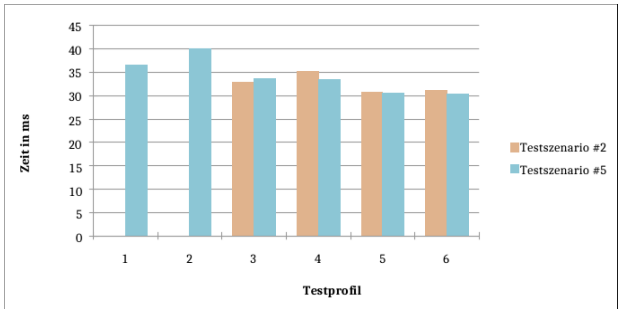
Grafik 5.59: Varianz, low, m.G.



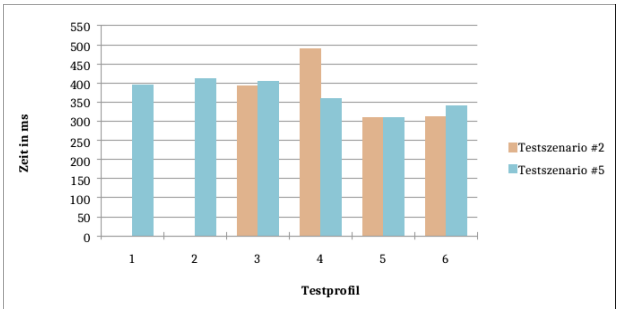
Grafik 5.60: Varianz, med, o.G.



Grafik 5.61: Varianz, med, m.G.



Grafik 5.62: Varianz, high, o.G.

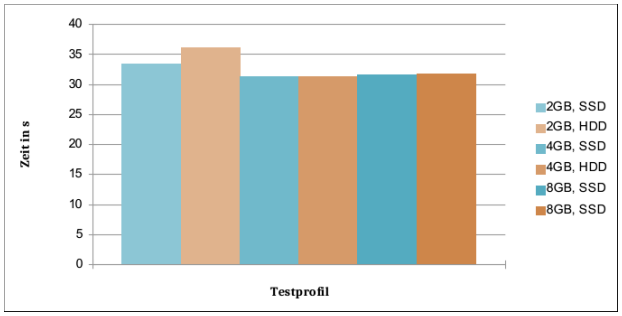


Grafik 5.63: Varianz, high, m.G.

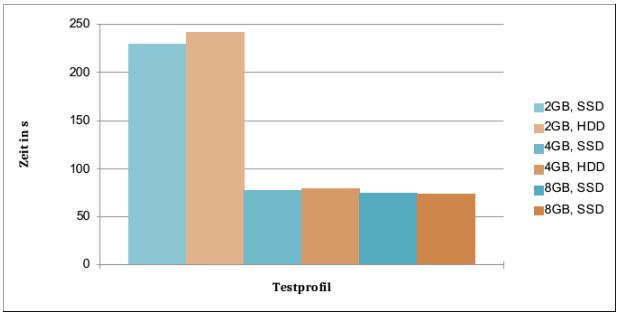
Bei der Varianz halten sich beide Testszenarien die Waage. Mal ist Testszenario #2 schneller in der Berechnung, mal Testszenario #5. Trotzdem kommt es hier zu einem erstaunlichen Ergebnis: Wie kann es sein, dass die Berechnungen in `PL/R` mit denen in `R` mithalten kann? In vielen Fällen sind die Berechnungen in `PL/R`, nimmt man die Mittelwerte her, sogar schneller. Fragen, die man so an dieser Stelle nicht beantworten kann.

Testszenario #6: R und DBI

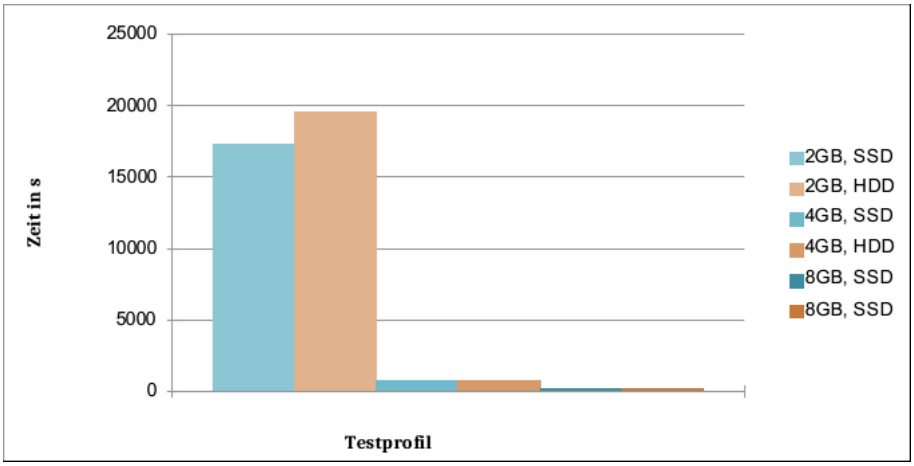
Wurden in den Testszenarien der Fall abgedeckt, dass `R` in `PgSQL` integriert ist, wird nun die Richtung umgedreht. Es wird aus `R` heraus auf `PgSQL` zugegriffen.



Grafik 5.64: kleines Datenpaket

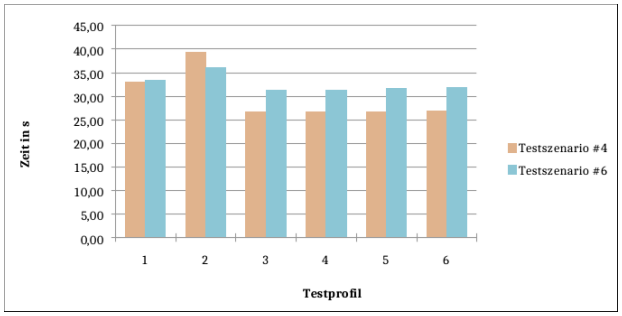


Grafik 5.65: mittleres Datenpaket

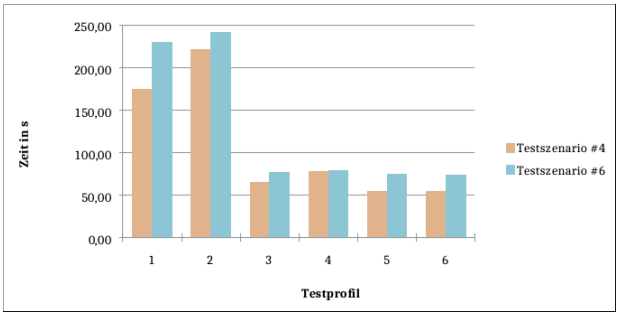


Grafik 5.66: großes Datenpaket

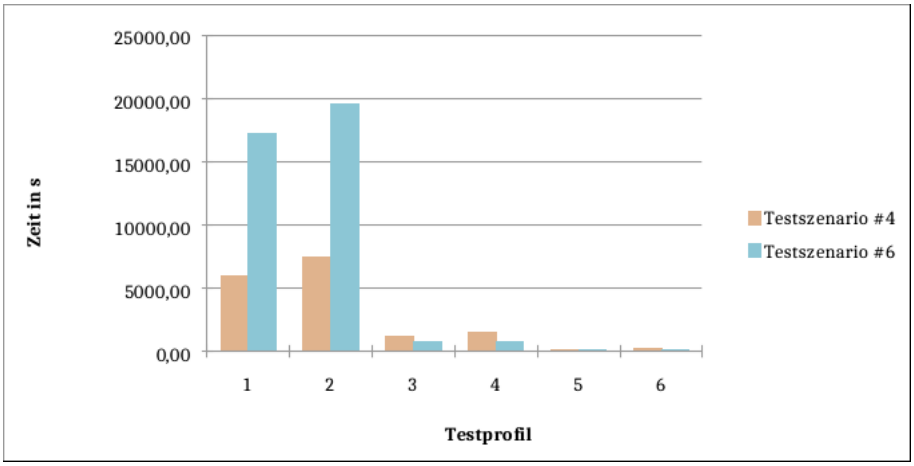
Die Grafiken offenbaren keine Überraschungen. In Grafik 5.64 sind die Einlesezeiten nahezu ausgeglichen. Beim mittleren Datenpaket in Grafik 5.65 tritt wieder das typische `R`-Problem zum Vorschein: Zu wenig Arbeitsspeicher, so dass auf den Swapbereich zugegriffen werden muss. Dabei machen sich auch hier wieder die unterschiedlichen Massenspeicher bemerkbar, denn SSD ist auch hier etwas schneller als HDD.



Grafik 5.67: kleines Datenpaket



Grafik 5.68: mittleres Datenpaket



Grafik 5.69: großes Datenpaket

Vergleicht man nun dieses Testszenario mit Testszenario #4, so muss man feststellen, dass Testszenario #4 in so gut wie jedem Testfall die Nase vorn hat. Lediglich beim kleinen Datenpaket in den 2GB-Testprofilen und beim großen Datenpaket in den 4GB-Testprofilen scheint es so, als ob sich das Bild drehen würde. Eine konkrete Aussage darüber kann allerdings an dieser Stelle nicht getroffen werden.

6. Zusammenfassung

Fasst man die Testergebnisse zusammen, muss deutlich gesagt werden, dass sich die Integration von R in einer Datenbank (hier PgSQL) lohnen kann. Gerade in Testszenario #3 wurde gezeigt, dass das Einlesen der Daten innerhalb von PgSQL einen großen Vorteil hat: Wo die reine R-Umgebung innerhalb von 12 Stunden nicht in der Lage war auch nur ein Testergebnis zu generieren, konnte PL/R überzeugen. Die Einlesezeit schoss zwar extrem nach oben, weshalb man nicht unbedingt von effizient sprechen kann, trotzdem kam, und da muss man an dieser Stelle wiederholen, ein Testergebnis raus. Zudem zeigte Testszenario #5, dass die statistischen Berechnungen innerhalb von PgSQL und PL/R nicht länger dauern als in einer reinen R-Umgebung. Zum Teil war PL/R sogar schneller. Testszenario #4 betrachtete den Zugriff aus PL/R auf die Datenbank selbst und kam zu einem ordentlichen Ergebnis. Auch hier konnten die 2GB-Testprofile die großen Datenpakete einlesen. In Punkto Geschwindigkeit muss sich `pg.spi.exec()` allerdings `fread()` mal mehr, mal weniger deutlich geschlagen geben. Trotzdem verliert die Kopplung von R mit Datenbankzugriff nicht den Reiz, was Testszenario #6 zeigte. Auch wenn die Einlesezeiten stets langsamer waren als in Testszenario #4, sind keine großen Unterschiede aufgedeckt worden.

Um die Frage dieser Bachelorarbeit zu klären, ob es sinnvoll ist R und SQL zu koppeln, so kommt man an dieser Stelle zu einem deutlich positiven Ergebnis. Denn die statistischen Möglichkeiten und die vielfältige Bibliothekenlandschaft von R sind einfach zu groß, als das man diese nicht mit einer Datenbank koppeln sollte. Nach den von dieser Bachelorarbeit durchgeführten Testszenarien spricht jedenfalls nichts dagegen.

7. Ausblick

Die vorliegende Arbeit wirft trotz des eindeutigen Ergebnisses einige Fragen auf, die durchaus Bestandteil weiterführender Arbeiten sein können. Da wäre als erstes die Frage, wieso PL/R in der Lage war, bei den 2GB-Testprofilen das große Datenpaket einzulesen und die reine R-Umgebung nicht. Kommt vielleicht R doch zu einem Ergebnis, wenn man das Zeitfenster für das Einlesen schlicht erweitert?
PL/R hat allerdings noch eine weitere spannende Frage aufkommen lassen, die so bisher noch nicht behandelt wurde: Wie wird R in den jeweiligen Datenbanken integriert und führen die DBMS-Anbieter Optimierungen durch (und wenn ja, welche?)? Denn die Ergebnisse lassen vermuten, dass es wohl (mindestens) einen weiteren Weg gibt, um Daten mit R-Syntax einzulesen, bzw. mit diesen zu arbeiten. Inwiefern dieser beschrieben werden kann, bleibt in dieser Arbeit ungeklärt und sollte ein spannendes Thema für andere Arbeiten sein.
Da diese Arbeit nur einfache statistische Berechnungen betrachtet, könnte sich ein anderes Forschungsprojekt mit komplexen Berechnungen auseinander setzen und diese, wie es die vorliegende Bachelorarbeit bereits tat, in reiner R-Umgebung, in reiner Datenbankumgebung und in Kopplung betrachten.

Literaturverzeichnis

[Cha14]: John M. Chambers, *Object-Oriented Programming, Functional Programming and R. In: Statistical Science*, Band 29, Nr. 2, 2014

[Con19]: Joseph Conway, *PL/R User's Guide - R Procedural Language*, 2019, Boston, USA, http://www.joeconway.com/plr/doc/plr-US.pdf, zuletzt aufgerufen am: 10.08.2019

[DBI-1]: R-DIG-DB, *Package 'DBI'*, 2018, , https://cran.r-project.org/web/packages/DBI/DBI.pdf, zuletzt aufgerufen am 16.08.2019

[IBM-1]: IBM Knowledge Center, *Db2 SQL: Built-in functions: Aggregate functions: MEDIAN*, 2019, https://www.ibm.com/support/knowledgecenter/en/SSEPEK_11.0.0/sqlref/src/tpc/db2z_bif_median.html, zuletzt aufgerufen am: 10.08.2019

[IBM-2]: IBM Knowledge Center, *Db2 SQL: Built-in functions: Aggregate functions: CUME_DIST*, 2019, https://www.ibm.com/support/knowledgecenter/SSCJDQ/com.ibm.swg.im.dashdb.sql.ref.doc/doc/r0062050.html, zuletzt aufgerufen am: 10.08.2019

[Mic-1]: Microsoft SQL-Dokumentaion, *Schnellstart: "Hello World" R-Skript in SQL Server*, 2019, https://docs.microsoft.com/de-de/sql/advanced-analytics/tutorials/quickstart-r-run-using-tsql?view=sql-server-2017, zuletzt aufgerufen am: 10.08.2019

[Mic-2]: Microsoft SQL Dokumentation, *Sp_execute_external_script (Transact-SQL)*, 2018, https://docs.microsoft.com/de-de/sql/relational-databases/system-stored-procedures/sp-execute-external-script-transact-sql?view=sql-server-2017, zuletzt aufgerufen am: 10.08.2019

[MoE02]: David Mosberger, Stephane Eranien, *IA-64 Linux Kernel: Design and Implementation*, 2002, Prentice Hall-Verlag,

[Mys-1]: Oracle Corporation, *MySQL 8.0 Reference Manual -> Window Function Descriptions*, 2019, https://dev.mysql.com/doc/refman/8.0/en/window-function-descriptions.html, zuletzt aufgerufen am: 10.08.2019

[Mys-2]: MySQLTutorial.org, *MySQL CUME_DIST Function*, 2019, http://www.mysqltutorial.org/mysql-window-functions/mysql-cume_dist-function/, zuletzt aufgerufen am: 10.08.2019

[Ora-1]: Oracle Help Center, *Database SQL Reference: CUME_DIST*, 2019, https://docs.oracle.com/cd/B19306_01/server.102/b14200/functions035.htm, zuletzt aufgerufen am: 10.08.2019

[Ora-2]: Mark Hornick, *Session 4: Oracle R Enterprise 1.4 Embedded R Execution -SQL*, 2014, https://www.oracle.com/technetwork/database/database-technologies/r/r-enterprise/learnmore/ore-1-4-embedded-r-execution-sql-2159066.pdf, zuletzt aufgerufen am: 10.08.2019

[Pos-1]: PostgreSQL Documentation, *Documentation -> PostgreSQL 11 -> Functions Window*, 2019, https://www.postgresql.org/docs/current/functions-window.html, zuletzt aufgerufen am: 10.08.2019

[Pos-2]: PostgreSQL Documentation, *Documentation -> PostgreSQL 11 -> CREATE FUNCTION*, 2019, https://www.postgresql.org/docs/current/sql-createfunction.html, zuletzt aufgerufen am: 15.08.2019

[Rct19]: R Core Team, *R Data Import/Export*, 2019, https://cran.r-project.org/doc/manuals/r-release/R-data.html, zuletzt aufgerufen am: 10.08.2019

[Sap-1]: SAP SE, *SAP HANA R Integration Guide*, 2018, https://help.sap.com/doc/6f2ff4c50f7e4e4d90b93aa33652d063/2.0.03/en-US/SAP_HANA_R_Integration_Guide_en.pdf, zuletzt aufgerufen am: 10.08.2019

[Sch16]: Maik Schmude, *Systematische Untersuchung der Anfragekapazität verschiedener DBMS*, Bachelorarbeit, Universität Rostock, Institut für Informatik, 2016, Rostock,

[Sha15]: Amit Sharma, *Cumulative Distribution Plots For Frequency Data in R*, 2015, , http://www.amitsharma.in/post/cumulative-distribution-plots-for-frequency-data-in-r/, zuletzt aufgerufen am: 10.08.2019

[Sha17]: Tomer Shay, *How to calculate median value in MySQL using a simple SQL Query*, 2017, https://www.eversql.com/how-to-calculate-median-value-in-mysql-using-a-simple-sql-query/, zuletzt aufgerufen am: 10.08.2019

[SSH18]: Gunter Saake, Kai-Uwe Sattler, Andreas Heuer, *Datenbanken: Konzepte und Sprachen*, 2018, Frechen, mitp Verlags GmbH & Co. KG, 6. Auflage

[Tec19]: TechOnTheNet, *Oracle / PLSQL: MEDIAN Function, 2019*, https://www.techonthenet.com/oracle/functions/median.php, zuletzt aufgerufen am: 10.08.2019

[VSR19]: W. N. Venables, D.M. Smith and the R Core Team, *An Introduction to R*, 2019, Version 3.6.1

Anhang

Testergebnisse

Testszenario #1: Einlesen von Daten

#	TP#1, low		TP#2, low	
	R	SQL	R	SQL
1	21,17	183,99	25,43	186,63
2	19,98	185,06	20,10	185,70
3	20,00	186,26	20,03	186,60
4	20,18	183,66	20,02	187,17
5	20,16	186,80	20,12	185,36
6	20,16	184,00	20,14	189,12
7	20,15	187,22	19,91	186,41
8	20,00	184,50	19,91	185,48
9	20,16	185,27	19,97	185,53
10	19,99	184,92	19,95	188,25
MW	20,20	185,17	20,56	186,63
%	816,68		807,73	

Tabelle A.1: Testszenario #1: 2GB, low

#	TP#1, med		TP#2, med	
	R	SQL	R	SQL
1	65,38	365,59	73,42	373,90
2	69,54	369,84	75,05	378,91
3	69,31	367,30	74,90	378,62
4	69,81	367,03	74,99	377,10
5	70,18	371,09	75,80	373,56
6	71,37	367,86	72,48	373,41
7	68,11	369,06	71,12	373,84
8	69,12	370,77	72,64	374,97
9	70,53	367,95	74,59	374,74
10	70,75	371,20	73,86	376,88
MW	69,41	368,77	73,89	375,59
%	431,29		408,31	

Tabelle A.2: Testszenario #1: 2GB, med

#	TP#1, high		TP#2, high	
	R	SQL	R	SQL
1	*	744,27	*	747,21
2	*	744,30	*	741,23
3	*	742,76	*	743,44
4	*	739,33	*	742,61
5	*	744,45	*	745,58
6	*	742,81	*	746,09
7	*	743,91	*	741,52
8	*	737,50	*	742,61
9	*	737,93	*	741,20
10	*	742,89	*	741,62
MW	*	742,02	*	743,31
%	*		*	

Tabelle A.3: Testszenario #1: 2GB, high

#	TP#3, low		TP#4, low	
	R	SQL	R	SQL
1	21,15	184,58	20,27	186,83
2	19,90	183,90	19,75	183,27
3	19,94	183,39	19,84	184,09
4	19,92	184,24	19,89	184,42
5	19,99	184,51	19,82	183,15
6	19,93	183,74	19,89	183,94
7	19,95	185,79	19,82	182,50
8	19,92	185,66	19,81	185,69
9	19,86	182,70	19,64	183,88
10	19,93	183,21	19,69	185,16
MW	20,05	184,17	19,84	184,29
%	818,55		828,88	

Tabelle A.4: Testszenario #1: 4GB, low

#	TP#3, med		TP#4, med	
	R	SQL	R	SQL
1	49,65	368,18	54,07	376,52
2	47,75	363,70	47,58	372,11
3	47,84	361,34	47,66	370,52
4	47,90	364,92	47,55	370,35
5	47,99	362,57	47,65	370,40
6	47,99	364,19	47,58	371,05
7	47,84	365,37	47,70	370,60
8	47,92	365,05	47,77	372,46
9	48,16	363,73	47,65	373,07
10	48,01	366,15	47,65	370,05
MW	48,11	364,52	48,29	371,71
%	657,68		669,75	

Tabelle A.5: Testszenario #1: 4GB, med

#	TP#3, high		TP#4, high	
	R	SQL	R	SQL
1	166,73	740,92	205,19	749,78
2	175,21	741,88	198,33	749,71
3	174,47	738,75	192,75	752,21
4	172,68	741,09	197,22	755,19
5	175,28	744,10	194,08	752,53
6	170,66	738,85	191,89	753,43
7	174,37	737,89	189,96	754,57
8	174,60	743,08	186,99	751,63
9	174,02	740,31	186,93	753,13
10	181,08	740,25	187,67	752,42
MW	173,91	740,71	193,10	752,46
%	325,92		289,67	

Tabelle A.6: Testszenario #1: 4GB, high

#	TP#5, low		TP#6, low	
	R	SQL	R	SQL
1	21,18	184,08	20,50	186,46
2	19,87	184,34	19,89	186,84
3	19,88	184,09	19,87	185,13
4	19,81	182,74	19,90	185,27
5	19,86	183,78	19,90	185,14
6	19,86	183,06	19,93	187,71
7	19,85	182,98	19,88	184,79
8	19,82	183,28	19,89	184,73
9	19,89	183,21	19,87	184,75
10	19,83	182,40	20,13	183,50
MW	19,98	183,40	19,98	185,43
%	817,92		828,08	

Tabelle A.7: Testszenario #1: 8GB, low

#	TP#5, med		TP#6, med	
	R	SQL	R	SQL
1	49,00	365,99	48,79	374,99
2	47,98	367,34	48,25	369,32
3	48,12	372,82	48,39	366,16
4	48,11	370,62	48,30	368,02
5	48,00	372,92	48,33	370,32
6	48,18	368,87	48,19	369,02
7	48,13	366,46	48,29	368,97
8	48,15	371,68	48,40	366,92
9	48,10	369,04	48,32	368,21
10	48,12	373,31	48,35	368,63
MW	48,19	369,91	48,36	369,06
%	667,61		663,15	

Tabelle A.8: Testszenario #1: 8GB, med

#	TP#5, high		TP#6, high	
	R	SQL	R	SQL
1	124,18	740,22	129,84	750,02
2	120,12	749,80	122,79	737,67
3	120,72	745,47	123,21	734,76
4	120,60	744,49	123,32	741,07
5	120,43	748,44	123,29	737,55
6	120,62	745,85	122,94	738,84
7	120,61	742,45	121,11	737,71
8	120,67	752,75	121,05	744,85
9	120,74	749,73	121,22	743,00
10	120,93	740,23	120,81	740,84
MW	129,96	745,94	122,96	740,63
%	473,98		502,33	

Tabelle A.9: Testszenario #1: 8GB, high

Testszenario #2: Statistische Berechnungen

Mittelwert

#	TP#1, low, ohne Gruppierung		TP#1, low, mit Gruppierung	
	R	SQL	R	SQL
1	4,40	1069,06	64,19	1332,91
2	4,27	1011,74	65,05	1332,00
3	4,27	1110,36	65,40	1383,51
4	4,27	1090,07	65,44	1363,64
5	4,28	1007,83	65,25	1300,38
6	4,27	1087,20	64,06	1402,38
7	4,27	1100,85	63,75	1289,33
8	4,27	1100,20	64,57	1304,22
9	4,41	1102,39	63,86	1404,13
10	4,36	1033,46	64,24	1365,36
MW	4,31	1071,32	64,58	1347,79
%	24756,61		1987,01	

Tabelle A.10: Testszenario #2: Mittelwert, 2GB, SSD, low

#	TP#2, low, ohne Gruppierung		TP#2, low, mit Gruppierung	
	R	SQL	R	SQL
1	4,34	1333,64	63,65	2096,04
2	4,42	1431,49	64,21	1932,42
3	4,44	2457,05	63,91	2029,47
4	4,26	1430,74	64,04	2158,19
5	4,29	1293,24	63,93	1720,78
6	4,47	1552,68	63,93	1953,87
7	4,31	1349,57	64,64	1659,34
8	4,28	1304,87	64,25	2241,09
9	4,28	1840,67	63,71	1940,30
10	4,26	1068,92	63,87	1867,24
MW	4,34	1506,29	64,01	1959,87
%	34607,14		2961,82	

Tabelle A.11: Testszenario #2: Mittelwert, 2GB, HDD, low

#	TP#3, low, ohne Gruppierung		TP#3, low, mit Gruppierung	
	R	SQL	R	SQL
1	4,28	797,41	70,82	1077,98
2	4,37	767,78	64,20	1101,89
3	4,36	818,26	65,07	1100,04
4	4,37	795,61	67,16	1078,20
5	4,42	822,35	65,25	1075,56
6	4,31	806,02	64,33	1081,22
7	4,26	822,93	63,60	1103,07
8	4,39	822,51	63,62	1128,37
9	4,38	795,34	63,86	1178,98
10	4,28	800,81	63,89	1105,88
MW	4,34	804,90	65,18	1103,12
%	18446,08		1592,42	

Tabelle A.12: Testszenario #2: Mittelwert, 4GB, SSD, low

#	TP#4, low, ohne Gruppierung		TP#4, low, mit Gruppierung	
	R	SQL	R	SQL
1	4,35	886,31	66,67	1315,54
2	4,44	813,06	63,64	1131,80
3	4,27	824,11	63,88	1111,85
4	4,25	893,55	64,00	1105,16
5	4,25	851,11	64,22	1101,86
6	4,28	821,09	64,41	1111,37
7	4,27	823,33	63,81	1131,01
8	4,34	845,31	64,27	1135,00
9	4,39	821,87	63,82	1102,64
10	4,27	821,00	63,82	1150,68
MW	4,31	840,07	64,25	1139,69
%	19391,18		1673,84	

Tabelle A.13: Testszenario #2: Mittelwert, 4GB, HDD, low

#	TP#5, low, ohne Gruppierung		TP#5, low, mit Gruppierung	
	R	SQL	R	SQL
1	4,27	749,80	67,28	961,24
2	4,34	759,06	63,70	769,09
3	4,36	751,70	64,26	770,07
4	4,25	752,34	63,41	765,11
5	4,36	753,80	64,72	767,62
6	4,28	755,56	63,37	771,43
7	4,39	750,20	64,04	768,48
8	4,32	753,38	63,85	771,43
9	4,25	751,23	64,30	766,69
10	4,35	752,17	63,41	767,25
MW	4,32	752,92	64,23	787,84
%	17328,7		1126,59	

Tabelle A.14: Testszenario #2: Mittelwert, 8GB, SSD, low

#	TP#6, low, ohne Gruppierung		TP#6, low, mit Gruppierung	
	R	SQL	R	SQL
1	4,25	832,44	68,20	1173,00
2	4,26	792,13	64,77	1079,58
3	4,36	797,65	64,65	1072,52
4	4,33	777,33	63,62	1116,90
5	4,34	789,61	64,19	1052,44
6	4,46	778,09	63,87	1108,68
7	4,34	788,99	63,89	1049,27
8	4,37	773,15	64,50	1138,63
9	4,29	783,04	64,60	1070,26
10	4,40	790,27	64,04	1055,44
MW	4,34	790,27	64,63	1091,67
%	18108,99		1589,11	

Tabelle A.15: Testszenario #2: Mittelwert, 8GB, HDD, low

#	TP#1, med, ohne Gruppierung		TP#1, med, mit Gruppierung	
	R	SQL	R	SQL
1	245,82	2562,56	709,40	3067,80
2	174,77	2587,60	577,21	3163,36
3	179,01	2613,25	552,44	3104,49
4	188,34	2587,52	517,94	3148,99
5	193,38	2643,58	514,95	3157,76
6	187,83	2616,15	659,28	3139,96
7	187,24	2624,88	554,55	3171,10
8	181,82	2604,77	575,41	3146,31
9	188,92	2599,44	531,81	3142,40
10	200,68	2554,96	571,57	3137,99
MW	192,78	2599,47	576,46	3138,02
%	1248,41		444,36	

Tabelle A.16: Testszenario #2: Mittelwert, 2GB, SSD, med

#	TP#2, med, ohne Gruppierung		TP#2, med, mit Gruppierung	
	R	SQL	R	SQL
1	311,43	5767,98	1201,32	6612,72
2	260,63	5859,25	1029,26	6297,12
3	352,34	9868,21	973,86	5895,82
4	268,26	5681,02	1006,75	6043,98
5	317,30	5395,62	1062,80	5929,71
6	272,34	5187,94	895,43	6268,77
7	388,89	5372,36	1024,72	6125,81
8	363,32	5527,72	941,37	6937,74
9	261,85	5275,33	952,93	6110,87
10	345,03	5373,80	1170,54	6059,89
MW	314,14	5930,92	1025,92	6228,24
%	1787,99		507,09	

Tabelle A.17: Testszenario #2: Mittelwert, 2GB, HDD, med

#	TP#3, med, ohne Gruppierung		TP#3, med, mit Gruppierung	
	R	SQL	R	SQL
1	8,37	1884,78	130,48	2495,85
2	8,60	1871,56	124,82	2415,54
3	8,46	1853,17	124,63	2552,78
4	8,27	1834,21	124,92	2429,80
5	8,74	1831,73	125,64	2410,70
6	8,35	1819,97	124,95	2466,81
7	8,33	1829,33	124,80	2459,60
8	8,41	1823,29	124,85	2451,44
9	8,80	1825,21	124,78	2470,92
10	8,38	1855,97	124,88	2399,48
MW	8,47	1842,92	125,48	2455,29
%	21658,21		1856,72	

Tabelle A.18: Testszenario #2: Mittelwert, 4GB, SSD, med

#	TP#4, med, ohne Gruppierung		TP#4, med, mit Gruppierung	
	R	SQL	R	SQL
1	8,28	2143,52	129,32	3245,74
2	8,33	1830,45	124,68	2814,75
3	8,35	1818,39	125,10	2523,84
4	8,35	1825,70	125,75	2655,63
5	8,43	1808,26	125,12	2608,73
6	8,85	1815,92	125,35	2456,95
7	8,34	1949,03	124,97	2466,46
8	8,26	1845,91	125,57	2426,77
9	8,44	1815,66	125,68	2473,13
10	8,34	2033,02	125,15	2596,18
MW	8,40	1888,59	125,67	2626,82
%	22383,21		1990,25	

Tabelle A.19: Testszenario #2: Mittelwert, 4GB, HDD, med

#	TP#5, med, ohne Gruppierung		TP#5, med, mit Gruppierung	
	R	SQL	R	SQL
1	8,35	1572,64	134,07	1969,80
2	8,64	1571,46	124,02	1500,16
3	8,22	1571,90	124,20	1511,52
4	8,34	1556,80	125,11	1506,84
5	8,26	1546,85	124,23	1523,39
6	8,36	1616,58	124,18	1522,31
7	8,32	1580,42	126,12	1523,26
8	8,34	1556,35	126,42	1505,80
9	8,37	1609,71	124,48	1504,18
10	8,40	1552,06	125,97	1510,12
MW	8,36	1573,48	125,88	1557,74
%	18721,53		1137,48	

Tabelle A.20: Testszenario #2: Mittelwert, 8GB, SSD, med

#	TP#6, med, ohne Gruppierung		TP#6, med, mit Gruppierung	
	R	SQL	R	SQL
1	8,35	1803,11	130,08	2322,44
2	8,37	1622,25	126,10	2247,41
3	8,32	1620,49	125,17	2228,38
4	8,25	1622,75	124,46	2363,06
5	8,32	1630,25	124,61	2221,65
6	8,37	1635,45	125,19	2222,37
7	8,32	1632,32	124,66	2239,63
8	8,37	1635,18	125,45	2244,59
9	8,51	1634,78	124,96	2235,97
10	8,34	1665,68	126,73	2280,00
MW	8,35	1650,23	125,74	2260,55
%	19663,23		1697,80	

Tabelle A.21: Testszenario #2: Mittelwert, 8GB, HDD, med

#	TP#1, high, ohne Gruppierung		TP#1, high, mit Gruppierung	
	R	SQL	R	SQL
1	*	3128,03	*	4547,58
2	*	3127,92	*	4417,18
3	*	3110,38	*	4415,36
4	*	3132,28	*	4038,37
5	*	3039,93	*	4059,96
6	*	3029,20	*	4083,18
7	*	3047,43	*	4329,77
8	*	3102,10	*	4410,19
9	*	3133,91	*	4503,01
10	*	3117,57	*	4530,46
MW	*	3096,88	*	4333,51
%	*		*	

Tabelle A.22: Testszenario #2: Mittelwert, 2GB, SSD, high

#	TP#2, high, ohne Gruppierung		TP#2, high, mit Gruppierung	
	R	SQL	R	SQL
1	*	3693,69	*	4664,73
2	*	3321,44	*	4965,52
3	*	3457,51	*	4239,94
4	*	3289,83	*	4247,56
5	*	3329,55	*	4426,86
6	*	3319,79	*	4499,44
7	*	3351,17	*	4414,92
8	*	3336,61	*	4412,30
9	*	3262,56	*	4427,08
10	*	3320,04	*	4603,34
MW	*	3368,22	*	4490,17
%	*		*	

Tabelle A.23: Testszenario #2: Mittelwert, 2GB, HDD, high

#	TP#3, high, ohne Gruppierung		TP#3, high, mit Gruppierung	
	R	SQL	R	SQL
1	704,88	3128,25	1414,79	4280,95
2	793,37	3076,54	1759,81	4281,13
3	726,20	3058,30	1751,85	4234,75
4	774,55	3137,94	1583,67	4243,58
5	781,19	3139,19	1493,82	4285,42
6	780,54	3148,02	1456,62	4247,18
7	782,37	3104,16	1607,52	4231,36
8	723,19	3081,56	1611,56	4246,14
9	773,31	3067,06	1586,33	4229,03
10	770,39	3068,48	1693,78	4260,34
MW	761,00	3100,95	1595,98	4253,99
%	307,48		166,54	

Tabelle A.24: Testszenario #2: Mittelwert, 4GB, SSD, high

#	TP#4, high, ohne Gruppierung		TP#4, high, mit Gruppierung	
	R	SQL	R	SQL
1	1244,33	3375,93	1706,34	4524,99
2	970,05	3371,13	1434,18	4500,83
3	844,54	3394,89	1457,16	4540,07
4	678,81	3356,95	1495,09	4526,84
5	796,30	3390,51	1560,64	4452,73
6	736,43	3393,30	1431,11	4518,33
7	869,39	3373,88	1395,39	4497,11
8	909,34	3361,69	1452,06	4532,46
9	961,50	3399,98	1414,51	4540,04
10	816,11	3381,49	1440,08	4495,98
MW	882,68	3379,98	1478,66	4512,94
%	113,91		103,91	

Tabelle A.25: Testszenario #2: Mittelwert, 4GB, HDD, high

#	TP#5, high, ohne Gruppierung		TP#5, high, mit Gruppierung	
	R	SQL	R	SQL
1	16,75	3099,67	261,85	4217,08
2	16,81	3041,04	242,85	4202,01
3	16,39	3060,16	241,49	4174,97
4	16,61	3068,12	242,13	4215,24
5	16,25	3051,45	241,90	4201,61
6	16,26	3072,54	242,09	4208,12
7	16,44	3045,11	243,31	4191,75
8	16,39	3054,36	242,34	4188,93
9	16,43	3045,95	241,60	4175,57
10	16,45	3050,59	242,22	4193,92
MW	16,48	3058,90	244,18	4196,92
%	18461,29		1618,78	

Tabelle A.26: Testszenario #2: Mittelwert, 8GB, SSD, high

#	TP#6, high, ohne Gruppierung		TP#6, high, mit Gruppierung	
	R	SQL	R	SQL
1	17,48	3252,44	268,61	4412,66
2	16,81	3270,11	244,89	4368,60
3	16,63	3256,09	246,04	4365,16
4	16,57	3257,82	245,51	4349,04
5	16,62	3317,15	247,38	4321,76
6	16,73	3298,49	245,62	4357,74
7	16,51	3302,63	243,59	4429,33
8	16,47	3285,43	242,16	4431,39
9	16,46	3333,48	241,68	4345,04
10	16,35	3325,06	243,48	4314,71
MW	16,66	3289,87	246,90	4369,54
%	19647,12		1669,76	

Tabelle A.27: Testszenario #2: Mittelwert, 8GB, HDD, high

Standardabweichung

#	TP#1, low, ohne Gruppierung		TP#1, low, mit Gruppierung	
	R	SQL	R	SQL
1	7,99	681,57	76,18	942,66
2	7,97	675,64	74,89	948,98
3	7,98	681,40	74,93	950,31
4	8,07	666,75	75,18	966,35
5	8,01	674,95	74,84	935,26
6	7,99	690,17	74,85	978,85
7	7,98	666,38	74,98	937,66
8	8,00	666,02	74,47	953,58
9	8,01	687,71	74,96	939,40
10	8,29	676,72	74,38	938,16
MW	8,03	676,73	74,97	949,12
%	8327,52		1166	

Tabelle A.28: Testszenario #2: Standardabweichung, 2GB, SSD, low

#	TP#2, low, ohne Gruppierung		TP#2, low, mit Gruppierung	
	R	SQL	R	SQL
1	7,89	857,37	82,16	1027,72
2	7,99	828,79	74,26	950,22
3	8,00	779,41	74,25	1038,93
4	8,04	924,07	74,78	981,13
5	8,02	767,99	74,89	1032,75
6	7,98	699,55	75,21	1054,44
7	8,03	858,14	74,72	1018,73
8	7,96	960,38	74,95	1010,90
9	7,99	844,37	74,29	951,88
10	7,97	849,87	74,58	1028,64
MW	7,99	836,99	75,41	1009,53
%	10375,47		1238,72	

Tabelle A.29: Testszenario #2: Standardabweichung, 2GB, HDD, low

#	TP#3, low, ohne Gruppierung		TP#3, low, mit Gruppierung	
	R	SQL	R	SQL
1	7,95	685,43	80,32	944,92
2	8,10	664,38	75,24	930,70
3	7,88	667,25	76,31	933,20
4	7,96	668,30	75,35	930,34
5	8,07	668,68	75,36	930,41
6	7,98	677,94	75,72	933,27
7	7,97	667,63	74,87	936,56
8	7,94	666,74	75,96	958,41
9	8,07	667,58	75,11	930,14
10	8,01	671,10	75,55	945,40
MW	7,99	670,50	75,98	937,34
%	8291,74		1133,67	

Tabelle A.30: Testszenario #2: Standardabweichung, 4GB, SSD, low

#	TP#4, low, ohne Gruppierung		TP#4, low, mit Gruppierung	
	R	SQL	R	SQL
1	8,00	675,27	76,93	955,01
2	7,99	703,31	74,77	956,70
3	8,00	667,42	74,21	981,77
4	7,98	668,76	74,88	942,78
5	7,96	810,91	75,13	963,85
6	8,01	674,94	74,63	939,97
7	8,01	672,48	75,72	960,80
8	8,01	675,49	74,95	960,64
9	7,99	675,89	75,12	945,42
10	8,02	674,10	74,74	973,71
MW	8,00	689,86	75,11	958,07
%	8523,25		1175,56	

Tabelle A.31: Testszenario #2: Standardabweichung, 4GB, HDD, low

#	TP#5, low, ohne Gruppierung		TP#5, low, mit Gruppierung	
	R	SQL	R	SQL
1	8,02	656,79	79,02	952,38
2	8,02	656,70	74,06	923,43
3	7,89	658,43	75,36	922,62
4	7,97	660,93	74,59	925,03
5	7,98	642,21	74,81	925,64
6	7,99	660,08	74,08	922,21
7	8,15	662,10	74,75	923,14
8	7,90	662,96	74,22	922,35
9	8,04	663,02	74,65	919,77
10	7,87	664,28	75,02	923,49
MW	7,98	658,75	75,06	926,01
%	8155,01		1133,69	

Tabelle A.32: Testszenario #2: Standardabweichung, 8GB, SSD, low

#	TP#6, low, ohne Gruppierung		TP#6, low, mit Gruppierung	
	R	SQL	R	SQL
1	8,46	677,72	78,90	953,89
2	8,04	674,09	75,78	1109,67
3	8,03	665,82	74,36	953,72
4	7,99	663,68	74,50	1112,62
5	7,97	675,07	74,37	942,43
6	7,95	664,98	74,93	1106,82
7	8,12	680,40	74,66	941,71
8	8,20	667,61	74,08	1097,48
9	8,27	670,23	74,36	952,58
10	8,02	675,87	75,11	939,38
MW	8,11	671,55	75,11	1011,03
%	8180,52		1246,07	

Tabelle A.33: Testszenario #2: Standardabweichung, 8GB, HDD, low

#	TP#1, med, ohne Gruppierung		TP#1, med, mit Gruppierung	
	R	SQL	R	SQL
1	21,75	1355,33	219,43	1845,25
2	22,29	1322,50	186,86	1846,59
3	22,27	1324,95	185,14	1848,70
4	35,82	1326,32	196,87	1851,87
5	34,14	1385,49	184,42	1855,24
6	35,64	1335,62	179,84	1850,29
7	32,74	1351,36	174,64	1854,14
8	27,59	1350,69	200,47	1857,91
9	35,23	1329,09	205,40	1855,50
10	28,20	1327,86	190,33	1877,84
MW	29,57	1340,92	192,34	1854,33
%	4434,73		864,09	

Tabelle A.34: Testszenario #2: Standardabweichung, 2GB, SSD, med

#	TP#2, med, ohne Gruppierung		TP#2, med, mit Gruppierung	
	R	SQL	R	SQL
1	31,52	1442,22	293,13	2206,91
2	22,54	1578,28	330,48	1969,72
3	22,77	1316,65	290,52	2092,45
4	27,58	1331,51	280,39	1860,06
5	23,18	1380,28	333,85	2029,25
6	20,09	1465,32	289,63	1848,93
7	22,67	1440,34	334,45	1851,99
8	23,03	1437,79	356,04	1853,77
9	23,04	1431,75	293,10	2078,55
10	23,35	1380,10	316,62	1869,35
MW	23,98	1420,42	311,82	1966,10
%	5823,35		530,52	

Tabelle A.35: Testszenario #2: Standardabweichung, 2GB, HDD, med

#	TP#3, med, ohne Gruppierung		TP#3, med, mit Gruppierung	
	R	SQL	R	SQL
1	15,60	1330,60	157,43	1867,45
2	15,57	1303,81	150,49	1853,02
3	15,59	1295,07	150,64	1929,37
4	15,72	1297,12	150,86	1855,30
5	15,86	1300,24	150,19	1856,34
6	15,58	1296,99	152,32	1851,76
7	16,28	1312,37	151,22	1881,53
8	15,70	1296,28	151,00	1888,65
9	16,02	1300,71	151,68	1863,27
10	15,71	1302,26	150,10	1853,97
MW	15,76	1303,55	151,59	1870,07
%	8171,26		1133,64	

Tabelle A.36: Testszenario #2: Standardabweichung, 4GB, SSD, med

#	TP#4, med, ohne Gruppierung		TP#4, med, mit Gruppierung	
	R	SQL	R	SQL
1	15,84	1477,05	155,73	1981,85
2	15,67	1612,61	151,75	2198,15
3	15,69	1527,51	150,92	1963,12
4	15,52	1538,03	151,20	1910,47
5	15,74	1517,23	151,34	1998,99
6	15,56	1544,89	151,38	1972,30
7	15,68	1517,07	150,91	2025,89
8	15,95	1482,07	150,77	1952,31
9	15,68	1497,94	151,73	2157,33
10	15,52	1412,44	150,97	1930,52
MW	15,69	1512,68	151,67	2009,09
%	9541,05		1224,65	

Tabelle A.37: Testszenario #2: Standardabweichung, 4GB, HDD, med

#	TP#5, med, ohne Gruppierung		TP#5, med, mit Gruppierung	
	R	SQL	R	SQL
1	15,90	1299,76	160,06	1850,80
2	15,54	1289,23	149,81	1815,46
3	15,62	1306,75	150,49	1810,35
4	15,70	1308,15	150,69	1814,99
5	16,09	1294,98	150,63	1828,69
6	15,56	1318,22	150,61	1821,49
7	15,66	1294,81	149,82	1820,58
8	15,81	1298,78	149,65	1810,25
9	15,57	1314,45	152,13	1815,81
10	15,52	1299,86	151,19	1822,61
MW	15,70	1302,50	151,51	1821,10
%	8196,18		1101,97	

Tabelle A.38: Testszenario #2: Standardabweichung, 8GB, SSD, med

#	TP#6, med, ohne Gruppierung		TP#6, med, mit Gruppierung	
	R	SQL	R	SQL
1	15,81	1342,26	154,71	1840,47
2	15,53	1373,26	151,57	1833,23
3	15,98	1312,19	150,35	1834,19
4	15,71	1333,56	150,98	1840,18
5	15,69	1363,90	151,87	1840,23
6	15,72	1355,20	150,77	1846,50
7	15,50	1316,07	150,58	1848,34
8	15,51	1319,83	152,10	1879,44
9	15,75	1330,82	151,62	1924,25
10	15,87	1388,68	150,74	1904,28
MW	15,71	1343,58	151,53	1859,11
%	8452,39		1126,97	

Tabelle A.39: Testszenario #2: Standardabweichung, 8GB, HDD, med

#	TP#1, high, ohne Gruppierung		TP#1, high, mit Gruppierung	
	R	SQL	R	SQL
1	*	3817,43	*	4970,02
2	*	3958,75	*	5010,73
3	*	4004,92	*	5105,90
4	*	3959,17	*	5029,97
5	*	4168,31	*	5275,71
6	*	3906,64	*	5078,60
7	*	3953,75	*	5009,98
8	*	3996,39	*	5015,94
9	*	3910,01	*	5104,12
10	*	3930,80	*	4986,50
MW	*	3960,62	*	5058,75
%	*		*	

Tabelle A.40: Testszenario #2: Standardabweichung, 2GB, SSD, high

#	TP#2, high, ohne Gruppierung		TP#2, high, mit Gruppierung	
	R	SQL	R	SQL
1	*	4605,92	*	6014,06
2	*	3932,17	*	5109,47
3	*	4100,76	*	4982,71
4	*	4061,85	*	5201,32
5	*	3965,44	*	5103,95
6	*	3962,95	*	5135,09
7	*	3935,67	*	5069,36
8	*	3965,74	*	5069,12
9	*	3852,26	*	5032,49
10	*	3894,37	*	5117,33
MW	*	4027,71	*	5183,49
%	*		*	

Tabelle A.41: Testszenario #2: Standardabweichung, 2GB, HDD, high

#	TP#3, high, ohne Gruppierung		TP#3, high, mit Gruppierung	
	R	SQL	R	SQL
1	37,16	2585,99	404,88	3751,74
2	39,06	2566,70	413,55	3710,20
3	36,87	2600,91	418,98	3752,75
4	40,79	2613,23	414,39	3726,37
5	37,40	2590,35	439,80	3755,18
6	40,28	2669,43	453,29	3810,30
7	39,37	2603,70	458,33	3780,11
8	41,41	2594,42	443,38	3745,23
9	39,07	2615,35	428,16	3821,91
10	37,90	2597,05	455,74	3775,54
MW	38,93	2603,71	433,05	3762,93
%	6588,18		768,94	

Tabelle A.42: Testszenario #2: Standardabweichung, 4GB, SSD, high

#	TP#4, high, ohne Gruppierung		TP#4, high, mit Gruppierung	
	R	SQL	R	SQL
1	51,35	2788,10	697,16	3899,08
2	38,79	2478,44	431,30	3590,17
3	44,44	2485,58	519,35	3643,23
4	43,03	2485,23	575,24	3655,54
5	36,21	2481,83	454,60	3589,70
6	49,81	2482,20	481,74	3653,37
7	39,64	2468,97	547,28	3590,24
8	39,82	2469,84	583,91	3580,17
9	36,45	2460,84	630,04	3573,94
10	46,99	2501,72	349,00	3609,24
MW	42,65	2510,28	526,96	3638,47
%	5785,77		590,46	

Tabelle A.43: Testszenario #2: Standardabweichung, 4GB, HDD, high

#	TP#5, high, ohne Gruppierung		TP#5, high, mit Gruppierung	
	R	SQL	R	SQL
1	30,79	2626,49	323,67	3957,57
2	31,11	2585,54	308,16	3668,93
3	31,39	2632,90	307,87	3603,90
4	30,99	2594,39	307,50	3617,04
5	30,95	2676,92	307,35	3640,63
6	30,81	2593,88	307,34	3617,51
7	31,28	2606,59	306,18	3621,23
8	31,00	2602,11	308,45	3631,42
9	30,92	2610,79	307,06	3685,84
10	31,62	2602,82	307,94	3666,75
MW	31,09	2613,24	309,15	3671,08
%	8305.4		1087,48	

Tabelle A.44: Testszenario #2: Standardabweichung, 8GB, SSD, high

#	TP#6, high, ohne Gruppierung		TP#6, high, mit Gruppierung	
	R	SQL	R	SQL
1	32,53	3047,88	326,50	4073,95
2	31,26	2997,80	310,42	4015,00
3	31,27	3111,13	314,03	4013,93
4	32,39	3013,40	311,26	3958,58
5	31,22	3032,57	312,49	3966,89
6	31,20	3022,94	306,95	3980,22
7	31,35	3032,71	306,62	4001,37
8	31,08	3001,99	308,58	4018,37
9	31,05	3001,16	306,50	4044,35
10	31,85	3078,67	306,71	4039,15
MW	31,52	3034,03	311,00	4011,18
%	9525,73		1189,77	

Tabelle A.45: Testszenario #2: Standardabweichung, 8GB, HDD, high

Varianz

#	TP#1, low, ohne Gruppierung		TP#1, low, mit Gruppierung	
	R	SQL	R	SQL
1	8,20	661,90	82,79	929,94
2	7,87	665,43	74,64	930,14
3	7,93	665,36	74,64	934,86
4	7,87	650,11	75,67	941,70
5	8,07	665,34	74,32	932,55
6	7,79	666,82	76,32	933,11
7	7,86	653,37	73,99	921,97
8	7,81	655,21	74,68	938,13
9	7,75	671,77	74,08	925,14
10	8,19	663,18	75,17	925,60
MW	7,93	661,85	75,63	931,31
%	8246,15		1131,4	

Tabelle A.46: Testszenario #2: Varianz, 2GB, SSD, low

#	TP#2, low, ohne Gruppierung		TP#2, low, mit Gruppierung	
	R	SQL	R	SQL
1	7,80	669,13	80,11	937,11
2	8,05	675,83	74,15	932,52
3	8,32	686,41	74,73	942,59
4	7,85	677,63	74,12	944,35
5	7,79	665,24	74,70	927,98
6	7,97	680,26	74,57	956,67
7	7,85	672,93	74,16	946,23
8	7,97	666,59	74,17	937,27
9	7,86	687,40	74,47	927,44
10	7,78	690,75	75,04	928,51
MW	7,92	677,22	75,02	938,07
%	8450,76		1150,43	

Tabelle A.47: Testszenario #2: Varianz, 2GB, HDD, low

#	TP#3, low, ohne Gruppierung		TP#3, low, mit Gruppierung	
	R	SQL	R	SQL
1	7,90	664,52	79,67	931,84
2	7,70	651,94	75,87	918,21
3	7,97	656,58	74,92	917,00
4	7,87	654,76	75,67	916,72
5	7,70	655,23	75,72	920,53
6	7,79	663,90	74,73	923,00
7	7,81	655,33	74,85	918,52
8	7,83	655,44	75,31	947,12
9	7,72	656,76	73,97	918,45
10	7,81	656,00	74,51	930,89
MW	7,81	657,05	75,52	924,23
%	8312,93		1123,82	

Tabelle A.48: Testszenario #2: Varianz, 4GB, SSD, low

#	TP#4, low, ohne Gruppierung		TP#4, low, mit Gruppierung	
	R	SQL	R	SQL
1	7,74	664,81	76,14	937,08
2	7,72	653,96	75,28	925,68
3	7,82	657,39	74,97	928,11
4	7,82	695,12	75,10	968,30
5	7,80	756,46	74,13	947,12
6	7,86	656,83	74,81	975,36
7	7,91	659,64	74,14	924,37
8	7,72	663,50	74,48	926,36
9	7,73	652,35	74,68	930,62
10	7,86	658,00	74,12	940,23
MW	7,80	671,81	74,79	940,32
%	8512,95		1157,28	

Tabelle A.49: Testszenario #2: Varianz, 4GB, HDD, low

#	TP#5, low, ohne Gruppierung		TP#5, low, mit Gruppierung	
	R	SQL	R	SQL
1	7,85	644,04	77,91	1007,20
2	7,70	648,11	74,70	924,25
3	7,87	646,88	74,30	922,90
4	7,77	654,24	73,81	920,39
5	7,77	649,60	75,18	923,82
6	7,82	651,74	75,15	924,99
7	7,83	652,40	74,20	924,82
8	7,88	649,52	74,64	921,61
9	7,80	652,36	74,11	921,08
10	7,80	635,24	73,92	923,20
MW	7,81	648,41	74,79	931,43
%	8202,3		1145,39	

Tabelle A.50: Testszenario #2: Varianz, 8GB, SSD, low

#	TP#6, low, ohne Gruppierung		TP#6, low, mit Gruppierung	
	R	SQL	R	SQL
1	7,91	661,79	77,13	938,13
2	7,82	662,79	74,43	1076,42
3	7,82	650,20	74,74	942,43
4	7,82	654,45	74,90	1017,80
5	7,69	668,08	74,67	926,75
6	7,99	654,92	73,98	998,83
7	7,73	666,70	75,02	923,35
8	7,84	657,11	73,96	973,35
9	7,79	660,64	74,56	942,49
10	7,93	666,55	75,88	932,55
MW	7,83	660,32	74,93	967,21
%	8333,21		1190,82	

Tabelle A.51: Testszenario #2: Varianz, 8GB, HDD, low

#	TP#1, med, ohne Gruppierung		TP#1, med, mit Gruppierung	
	R	SQL	R	SQL
1	16,23	1292,30	155,97	1812,03
2	16,40	1299,29	177,42	1815,72
3	15,80	1305,43	177,36	1832,34
4	17,45	1299,37	219,06	1875,24
5	17,90	1325,78	175,26	1821,63
6	21,98	1299,55	202,59	1820,13
7	16,47	1343,34	174,83	1825,65
8	16,72	1315,09	205,81	1828,90
9	16,42	1305,97	180,12	1823,99
10	16,17	1300,65	196,08	1827,33
MW	17,15	1308,68	186,45	1828,30
%	7530,79		880,58	

Tabelle A.52: Testszenario #2: Varianz, 2GB, SSD, med

#	TP#2, med, ohne Gruppierung		TP#2, med, mit Gruppierung	
	R	SQL	R	SQL
1	21,41	1307,94	265,07	1809,45
2	16,92	1285,30	212,61	1829,17
3	16,40	1294,45	215,42	1814,72
4	17,10	1289,41	265,45	1836,44
5	16,47	1293,76	210,17	1820,11
6	15,86	1288,21	274,82	1820,56
7	16,10	1293,87	339,63	1816,68
8	17,30	1289,55	238,26	1820,65
9	16,17	1307,23	265,28	1834,69
10	16,18	1289,05	286,55	1816,40
MW	16,99	1293,88	257,33	1821,89
%	7515,54		608,00	

Tabelle A.53: Testszenario #2: Varianz, 2GB, HDD, med

#	TP#3, med, ohne Gruppierung		TP#3, med, mit Gruppierung	
	R	SQL	R	SQL
1	15,54	1282,03	151,60	1832,70
2	15,83	1266,83	148,06	1821,17
3	15,52	1263,92	147,57	1820,32
4	15,48	1265,24	147,05	1825,75
5	15,36	1283,64	147,43	1822,89
6	15,48	1270,17	148,28	1823,61
7	15,66	1288,65	147,87	1853,88
8	15,94	1270,26	148,11	1898,83
9	15,57	1269,15	147,03	1836,73
10	15,71	1267,88	148,37	1820,53
MW	15,61	1272,78	148,14	1835,64
%	8053,62		1139,13	

Tabelle A.54: Testszenario #2: Varianz, 4GB, SSD, med

#	TP#4, med, ohne Gruppierung		TP#4, med, mit Gruppierung	
	R	SQL	R	SQL
1	15,43	1287,79	151,23	1827,93
2	15,39	1283,61	148,98	1836,79
3	15,36	1283,93	148,39	1856,28
4	15,51	1300,69	148,53	1818,49
5	15,42	1279,36	147,60	1846,15
6	15,37	1281,34	148,19	1849,16
7	15,32	1300,77	147,94	1845,26
8	15,41	1283,04	147,36	1830,37
9	15,56	1298,25	148,48	1827,22
10	15,60	1306,93	147,65	1827,15
MW	15,44	1290,57	148,44	1836,48
%	8258,61		1137,19	

Tabelle A.55: Testszenario #2: Varianz, 4GB, HDD, med

#	TP#5, med, ohne Gruppierung		TP#5, med, mit Gruppierung	
	R	SQL	R	SQL
1	15,51	1275,55	155,34	1827,83
2	15,56	1265,06	147,17	1818,16
3	15,35	1282,48	147,32	1815,62
4	15,33	1294,07	146,81	1822,15
5	15,56	1269,66	148,13	1830,55
6	15,54	1284,63	147,53	1825,64
7	15,30	1270,42	147,18	1816,13
8	15,74	1272,49	147,01	1816,46
9	16,07	1293,87	147,72	1812,39
10	15,68	1271,44	146,94	1810,90
MW	15,56	1277,97	148,12	1819,58
%	8113,17		1128,45	

Tabelle A.56: Testszenario #2: Varianz, 8GB, SSD, med

#	TP#6, med, ohne Gruppierung		TP#6, med, mit Gruppierung	
	R	SQL	R	SQL
1	15,44	1270,54	151,85	1809,11
2	15,58	1284,75	148,33	1803,85
3	15,42	1268,10	148,49	1804,00
4	15,32	1266,59	148,49	1806,25
5	15,39	1267,89	147,70	1810,40
6	15,35	1281,51	149,47	1813,48
7	15,49	1267,89	150,55	1807,47
8	15,36	1271,51	147,77	1835,73
9	15,33	1268,11	148,04	1834,47
10	15,36	1286,64	147,82	1828,79
MW	15,40	1273,35	148,85	1815,36
%	8168,51		1119,59	

Tabelle A.57: Testszenario #2: Varianz, 8GB, HDD, med

#	TP#1, high, ohne Gruppierung		TP#1, high, mit Gruppierung	
	R	SQL	R	SQL
1	*	3791,05	*	4873,43
2	*	3732,32	*	4983,94
3	*	3961,42	*	4942,37
4	*	3794,58	*	4901,51
5	*	4073,89	*	5304,76
6	*	3837,36	*	4927,05
7	*	3846,33	*	4895,93
8	*	3868,45	*	4935,02
9	*	3786,57	*	4909,71
10	*	3979,33	*	4904,09
MW	*	3867,13	*	4957,78
%	*		*	

Tabelle A.58: Testszenario #2: Varianz, 2GB, SSD, high

#	TP#2, high, ohne Gruppierung		TP#2, high, mit Gruppierung	
	R	SQL	R	SQL
1	*	4448,54	*	5193,81
2	*	4024,93	*	5830,34
3	*	3888,75	*	4903,26
4	*	3968,91	*	5133,89
5	*	3936,89	*	5075,56
6	*	3965,02	*	5144,26
7	*	3922,20	*	5039,81
8	*	3909,01	*	5003,17
9	*	3896,57	*	5122,96
10	*	3867,97	*	5177,21
MW	*	3982,88	*	5162,43
%	*		*	

Tabelle A.59: Testszenario #2: Varianz, 2GB, HDD, high

#	TP#3, high, ohne Gruppierung		TP#3, high, mit Gruppierung	
	R	SQL	R	SQL
1	33,38	2501,13	423,21	3778,91
2	32,96	2637,45	430,42	3796,19
3	33,27	2680,51	380,04	3770,42
4	32,93	2521,61	397,10	3830,21
5	33,78	2665,70	364,39	3828,16
6	32,29	2653,97	375,73	3830,86
7	32,05	2530,59	363,03	3926,41
8	33,68	2662,26	343,77	3829,54
9	32,46	2685,08	432,26	3796,79
10	32,47	2688,13	408,14	3801,21
MW	32,93	2622,64	391,81	3818,87
%	7864,29		874,67	

Tabelle A.60: Testszenario #2: Varianz, 4GB, SSD, high

#	TP#4, high, ohne Gruppierung		TP#4, high, mit Gruppierung	
	R	SQL	R	SQL
1	33,78	2622,75	356,22	3836,22
2	33,40	2464,25	546,74	3581,08
3	33,42	2491,69	467,61	3610,41
4	33,30	2483,19	474,25	3642,46
5	32,38	2485,81	561,14	3580,72
6	31,28	2488,78	497,77	3599,84
7	36,55	2464,49	494,07	3598,22
8	33,43	2463,65	488,82	3571,26
9	33,36	2461,40	458,42	3569,99
10	50,72	2495,23	542,33	3606,13
MW	35,16	2492,12	488,74	3619,63
%	6987,94		640,60	

Tabelle A.61: Testszenario #2: Varianz, 4GB, HDD, high

#	TP#5, high, ohne Gruppierung		TP#5, high, mit Gruppierung	
	R	SQL	R	SQL
1	30,45	2812,94	325,46	3943,94
2	30,74	2542,27	310,87	3615,14
3	30,85	2788,57	305,89	3626,27
4	30,57	2539,84	306,39	3651,71
5	30,59	2546,89	306,01	3632,98
6	30,54	2827,36	306,82	3637,38
7	30,78	2827,88	307,07	3622,34
8	30,53	2828,29	306,94	3639,67
9	30,70	2787,03	306,20	3668,33
10	31,06	2551,59	307,06	3660,73
MW	30,68	2705,27	308,87	3669,85
%	8717,7		1088,15	

Tabelle A.62: Testszenario #2: Varianz, 8GB, SSD, high

#	TP#6, high, ohne Gruppierung		TP#6, high, mit Gruppierung	
	R	SQL	R	SQL
1	31,05	2540,77	324,97	3800,29
2	31,03	2511,24	313,52	3652,95
3	32,35	2571,15	310,31	3886,62
4	30,89	2515,29	314,18	3881,99
5	31,03	2501,02	311,13	3828,60
6	32,21	2540,60	307,52	3802,35
7	31,10	2549,74	309,21	3850,45
8	30,74	2516,29	306,49	3850,90
9	30,56	2518,21	306,87	3856,86
10	30,67	2552,16	308,03	3640,50
MW	31,16	2531,65	311,22	3805,15
%	8024,68		1122,66	

Tabelle A.63: Testszenario #2: Varianz, 8GB, HDD, high

Testszenario #3: PL/R und fread()

#	TP#1	TP#3	TP#5
1	20,96	20,91	20,96
2	19,77	19,63	19,71
3	19,78	19,60	19,71
4	19,75	19,68	19,74
5	19,77	19,60	19,76
6	19,85	19,71	19,73
7	19,76	19,67	19,75
8	19,82	19,63	19,75
9	19,79	19,81	19,81
10	19,80	19,74	19,72
MW	19,91	19,80	19,86

Tabelle A.64: Testszenario #3: SSD, low

#	TP#2	TP#4	TP#6
1	25,27	22,91	20,60
2	19,87	19,56	19,73
3	19,80	19,52	19,64
4	19,79	19,56	19,65
5	19,78	19,59	19,65
6	19,80	19,57	19,66
7	19,82	19,58	19,67
8	19,77	19,63	19,63
9	19,79	19,62	19,63
10	19,73	19,60	19,74
MW	20,34	19,91	19,76

Tabelle A.65: Testszenario #3: HDD, low

#	TP#1	TP#3	TP#5
1	69,06	50,42	50,27
2	67,04	48,01	48,36
3	68,99	48,06	48,79
4	69,26	48,15	48,76
5	69,20	47,97	48,56
6	67,39	48,02	48,71
7	67,52	47,98	48,70
8	69,03	48,20	48,74
9	68,78	48,06	48,65
10	68,53	48,07	48,73
MW	68,48	48,29	48,83

Tabelle A.66: Testszenario #3: SSD, med

#	TP#2	TP#4	TP#6
1	83,52	52,14	51,80
2	75,89	48,39	48,37
3	75,14	48,56	48,67
4	76,35	48,69	48,59
5	72,88	48,60	48,81
6	76,92	48,70	48,71
7	75,98	48,77	48,64
8	76,20	48,82	48,53
9	78,21	48,87	48,68
10	75,59	48,47	48,69
MW	76,67	49,00	48,95

Tabelle A.67: Testszenario #3: HDD, med

#	TP#1	TP#3	TP#5
1	5754,74	162,54	122,73
2	5772,75	168,76	120,05
3	5791,20	172,15	121,16
4	5778,95	170,99	121,59
5	5808,36	171,49	120,99
6	5742,83	162,09	121,17
7	5749,81	173,12	120,80
8	5766,09	172,96	121,41
9	5698,47	171,86	121,06
10	5757,65	172,72	120,73
MW	5762,09	169,87	121,17

Tabelle A.68: Testszenario #3: SSD, high

#	TP#2	TP#4	TP#6
1	10868,12	224,05	126,21
2	10833,79	194,59	121,75
3	10780,47	186,38	121,55
4	10878,03	204,30	120,98
5	10817,31	190,79	121,71
6	10868,87	195,57	121,41
7	10784,88	202,46	122,21
8	10859,33	188,26	121,66
9	10842,80	189,53	121,02
10	10803,56	200,44	121,88
MW	10833,72	197,64	122,04

Tabelle A.69: Testszenario #3: HDD, high

Testszenario #4: PL/R und pg.spi.exec()

#	TP#1	TP#3	TP#5
1	32,89	27,07	27,66
2	33,37	26,46	26,47
3	32,77	26,73	26,53
4	33,06	26,47	26,79
5	32,72	26,50	26,48
6	32,93	26,77	26,57
7	33,06	26,58	26,60
8	32,83	26,75	26,50
9	32,84	26,75	26,47
10	33,34	26,82	26,46
MW	32,98	26,69	26,65

Tabelle A.70: Testszenario #4: SSD, low

#	TP#2	TP#4	TP#6
1	40,33	27,39	28,73
2	38,99	26,56	26,44
3	39,47	26,39	26,39
4	38,97	26,43	26,64
5	38,95	26,51	26,55
6	39,43	26,44	26,83
7	39,72	26,51	26,45
8	39,18	26,59	26,50
9	39,34	26,79	26,60
10	39,33	26,59	26,56
MW	39,37	26,62	26,77

Tabelle A.71: Testszenario #4: HDD, low

#	TP#1	TP#3	TP#5
1	172,00	65,14	55,91
2	174,70	64,83	54,81
3	175,01	65,77	54,54
4	174,20	65,07	54,50
5	173,24	64,96	54,62
6	172,97	64,89	54,95
7	176,38	65,35	54,87
8	175,03	65,50	54,83
9	174,86	65,48	54,54
10	176,03	65,16	54,64
MW	174,44	65,22	54,82

Tabelle A.72: Testszenario #4: SSD, med

#	TP#2	TP#4	TP#6
1	219,27	79,84	56,34
2	230,94	78,67	54,67
3	221,00	77,19	54,17
4	222,76	77,12	54,59
5	222,55	77,42	54,69
6	220,13	77,44	55,74
7	220,82	77,13	54,42
8	220,49	77,46	54,48
9	218,55	76,56	55,34
10	219,45	77,02	55,08
MW	221,60	77,59	54,95

Tabelle A.73: Testszenario #4: HDD, med

#	TP#1	TP#3	TP#5
1	6048,65	1251,74	168,62
2	5939,07	1206,81	164,53
3	6052,08	1191,06	165,89
4	6039,06	1294,99	166,72
5	6057,05	1277,53	167,34
6	5986,83	1168,20	165,30
7	5948,47	1159,15	165,41
8	5984,27	1190,54	166,45
9	6094,30	1292,42	166,38
10	6077,84	1262,52	165,47
MW	6022,76	1229,50	166,21

Tabelle A.74: Testszenario #4: SSD, high

#	TP#2	TP#4	TP#6
1	7519,68	1515,29	250,44
2	7558,97	1458,33	247,13
3	7423,81	1386,85	233,22
4	7574,39	1562,33	228,66
5	7462,37	1478,98	219,83
6	7456,69	1580,80	214,89
7	7464,51	1558,96	226,91
8	7537,22	1435,51	220,14
9	7475,98	1496,62	225,75
10	7402,07	1536,49	229,54
MW	7487,57	1501,02	229,65

Tabelle A.75: Testszenario #4: HDD, high

Testszenario #5: PL/R und statistische Berechnungen

Mittelwert

#	TP#1	TP#3	TP#5
1	4,16	4,16	4,15
2	4,16	4,14	4,15
3	4,15	4,30	4,33
4	4,23	4,29	4,24
5	4,24	4,25	4,17
6	4,15	4,15	4,61
7	4,15	4,25	4,17
8	4,26	4,16	4,20
9	4,16	4,20	4,16
10	4,15	4,28	4,26
MW	4,18	4,22	4,24

Tabelle A.76: Testszenario #5: Mittelwert, SSD, low, o.G.

#	TP#2	TP#4	TP#6
1	4,26	4,28	4,16
2	4,16	4,24	4,43
3	4,16	4,26	4,16
4	4,15	4,15	4,17
5	4,16	4,60	4,15
6	4,23	4,53	4,15
7	4,36	4,16	4,16
8	4,30	4,17	4,29
9	4,24	4,27	4,16
10	4,15	4,15	4,17
MW	4,22	4,28	4,20

Tabelle A.77: Testszenario #5: Mittelwert, HDD, low, o.G.

#	TP#1	TP#3	TP#5
1	56,71	57,23	58,02
2	57,23	56,65	57,00
3	57,29	58,62	56,59
4	56,83	56,71	56,39
5	57,67	56,56	56,52
6	58,31	58,17	56,29
7	58,46	56,35	56,17
8	57,12	56,44	56,07
9	56,89	57,04	56,86
10	57,02	57,17	56,28
MW	57,35	57,09	56,62

Tabelle A.78: Testszenario #5: Mittelwert, SSD, low, m.G.

#	TP#2	TP#4	TP#6
1	57,11	56,95	56,52
2	57,04	56,33	56,56
3	56,68	56,13	56,86
4	56,16	56,05	56,57
5	56,77	56,80	57,92
6	57,24	56,24	56,34
7	58,01	57,21	56,12
8	56,34	57,09	57,19
9	56,30	58,04	56,36
10	57,19	56,99	56,34
MW	56,88	56,78	56,68

Tabelle A.79: Testszenario #5: Mittelwert, HDD, low, m.G.

#	TP#1	TP#3	TP#5
1	178,64	8,80	8,42
2	176,77	8,22	8,25
3	183,82	8,24	8,20
4	185,45	8,51	8,28
5	196,20	8,27	8,24
6	197,48	8,23	8,27
7	193,40	8,24	8,27
8	190,81	8,23	8,35
9	188,27	8,26	8,16
10	198,30	8,23	8,22
MW	188,91	8,32	8,27

Tabelle A.80: Testszenario #5: Mittelwert, SSD, med, o.G.

#	TP#2	TP#4	TP#6
1	192,76	8,45	8,22
2	193,81	8,24	8,24
3	199,26	8,12	8,25
4	198,34	8,22	8,24
5	192,75	8,21	8,29
6	195,34	8,25	8,44
7	199,73	8,24	8,16
8	192,56	8,29	8,32
9	199,50	8,23	8,19
10	199,96	8,23	8,12
MW	196,40	8,25	8,25

Tabelle A.81: Testszenario #5: Mittelwert, HDD, med, o.G.

#	TP#1	TP#3	TP#5
1	150,89	118,18	116,14
2	153,37	112,10	113,01
3	151,17	112,53	113,11
4	166,53	113,00	112,35
5	159,32	112,98	114,06
6	165,25	112,13	111,74
7	163,18	111,91	112,69
8	174,76	113,58	112,00
9	155,63	112,73	113,61
10	167,74	114,07	112,01
MW	160,78	113,32	113,07

Tabelle A.82: Testszenario #5: Mittelwert, SSD, med, m.G.

#	TP#2	TP#4	TP#6
1	246,49	115,32	121,52
2	225,84	112,27	114,46
3	168,32	112,90	112,93
4	145,45	112,57	112,93
5	131,64	114,15	113,09
6	130,86	114,49	112,15
7	202,91	112,48	113,43
8	195,82	112,93	112,78
9	175,86	113,25	113,66
10	226,01	112,92	112,43
MW	184,92	113,33	113,94

Tabelle A.83: Testszenario #5: Mittelwert, HDD, med, m.G.

#	TP#1	TP#3	TP#5
1	1539,97	391,17	16,19
2	1558,06	389,92	16,21
3	1585,98	404,23	16,23
4	1638,36	413,24	16,29
5	1543,97	386,88	16,28
6	1529,76	417,10	16,33
7	1586,88	394,86	16,24
8	1567,45	388,35	16,21
9	1593,29	420,66	16,27
10	1539,04	405,99	16,22
MW	1568,28	401,24	16,25

Tabelle A.84: Testszenario #5: Mittelwert, SSD, high, o.G.

#	TP#2	TP#4	TP#6
1	10320,03	409,06	16,23
2	10294,16	392,27	16,27
3	10326,87	423,23	16,30
4	10289,79	401,74	16,60
5	10367,02	412,87	16,02
6	10314,83	393,11	16,39
7	10313,87	429,46	16,19
8	10263,19	425,02	16,19
9	10269,68	416,73	16,79
10	10321,77	421,76	16,22
MW	10308,12	412,53	16,32

Tabelle A.85: Testszenario #5: Mittelwert, HDD, high, o.G.

#	TP#1	TP#3	TP#5
1	300,97	427,88	248,13
2	329,69	427,36	236,30
3	308,26	428,27	236,08
4	302,91	427,78	236,31
5	340,60	405,42	238,03
6	345,32	400,05	236,71
7	298,18	455,23	235,05
8	305,57	478,45	235,50
9	294,08	431,91	237,81
10	307,99	433,69	235,71
MW	313,36	431,60	237,56

Tabelle A.86: Testszenario #5: Mittelwert, SSD, high, m.G.

#	TP#2	TP#4	TP#6
1	10385,09	432,23	249,11
2	10343,77	352,70	237,36
3	10349,30	443,66	237,13
4	10337,06	444,68	234,03
5	10365,08	530,56	238,23
6	10388,97	311,77	237,63
7	10398,66	387,01	230,19
8	10380,98	492,14	239,12
9	10322,35	434,76	235,60
10	10376,04	375,64	232,68
MW	10364,73	420,52	237,11

Tabelle A.87: Mittelwert, Testszenario #5: HDD, high, m.G.

Standardabweichung

#	TP#1	TP#3	TP#5
1	7,95	7,78	7,77
2	7,70	7,78	7,77
3	7,77	7,68	7,72
4	7,67	7,76	7,77
5	7,78	7,81	7,78
6	7,84	7,79	7,88
7	7,77	7,78	7,85
8	7,64	7,69	7,78
9	7,76	7,77	7,78
10	7,76	7,83	7,69
MW	7,76	7,77	7,78

Tabelle A.88: Testszenario #5: Standardabweichung, SSD, low, o.G.

#	TP#2	TP#4	TP#6
1	7,82	7,79	7,80
2	7,84	7,83	7,76
3	8,35	7,66	7,81
4	7,82	7,86	7,80
5	7,79	7,88	7,79
6	7,79	8,06	7,79
7	7,70	7,79	7,79
8	7,70	7,79	7,68
9	7,76	7,69	7,76
10	7,81	8,58	7,77
MW	7,84	7,89	7,78

Tabelle A.89: Testszenario #5: Standardabweichung, HDD, low, o.G.

#	TP#1	TP#3	TP#5
1	77,19	74,42	74,91
2	75,10	74,36	73,89
3	75,79	74,03	75,12
4	74,71	74,29	74,36
5	75,13	74,43	73,82
6	75,17	74,38	74,03
7	75,20	75,27	74,46
8	76,40	73,99	76,22
9	74,90	75,26	73,88
10	74,89	74,23	73,81
MW	75,45	74,47	74,45

Tabelle A.90: Testszenario #5: Standardabweichung, SSD, low, m.G.

#	TP#2	TP#4	TP#6
1	74,46	74,22	77,38
2	74,20	74,43	75,25
3	75,38	74,95	73,99
4	74,82	74,38	73,96
5	75,00	75,66	75,01
6	74,28	74,41	74,79
7	74,41	74,25	74,87
8	73,98	76,18	74,80
9	74,23	74,99	73,94
10	74,38	74,97	74,41
MW	74,51	74,84	74,84

Tabelle A.91: Testszenario #5: Standardabweichung, HDD, low, m.G.

#	TP#1	TP#3	TP#5
1	30,49	15,93	15,59
2	20,50	16,05	15,49
3	22,66	15,23	15,29
4	27,25	16,47	15,52
5	20,19	15,28	15,60
6	24,98	15,50	15,44
7	22,74	15,45	15,32
8	28,61	15,32	15,43
9	28,79	16,00	15,47
10	20,21	15,43	15,51
MW	24,64	15,67	15,47

Tabelle A.92: Testszenario #5: Standardabweichung, SSD, med, o.G.

#	TP#2	TP#4	TP#6
1	30,23	15,62	15,32
2	23,40	15,25	15,42
3	21,30	15,32	15,48
4	25,87	15,26	15,27
5	22,83	15,27	15,44
6	25,57	15,52	15,75
7	26,04	15,44	15,45
8	20,71	15,36	15,35
9	20,73	15,57	15,46
10	20,43	15,30	15,46
MW	23,71	15,39	15,44

Tabelle A.93: Testszenario #5: Standardabweichung, HDD, med, o.G.

#	TP#1	TP#3	TP#5
1	192,44	172,18	152,87
2	198,15	148,12	148,15
3	189,02	147,57	148,10
4	186,98	148,91	147,24
5	188,96	147,71	147,54
6	181,26	147,91	147,82
7	186,57	147,94	147,71
8	189,54	148,19	147,69
9	196,97	147,63	148,65
10	198,00	148,32	148,01
MW	190,79	150,45	148,38

Tabelle A.94: Testszenario #5: Standardabweichung, SSD, med, m.G.

#	TP#2	TP#4	TP#6
1	200,07	151,47	159,73
2	293,51	148,57	150,21
3	205,05	148,31	148,98
4	263,09	149,52	147,78
5	269,37	148,86	148,61
6	264,89	148,66	148,53
7	195,46	148,63	147,89
8	206,24	148,21	148,69
9	220,01	153,07	147,52
10	221,14	149,52	148,58
MW	233,88	149,48	149,65

Tabelle A.95: Testszenario #5: Standardabweichung, HDD, med, m.G.

#	TP#1	TP#3	TP#5
1	46,85	36,60	30,47
2	50,11	40,43	30,60
3	49,51	37,24	30,60
4	44,54	47,22	30,61
5	52,71	47,10	30,67
6	42,91	40,11	30,64
7	44,57	36,87	30,64
8	47,12	37,15	30,65
9	51,20	41,25	30,61
10	49,64	43,66	30,62
MW	47,92	40,76	30,61

Tabelle A.96: Testszenario #5: Standardabweichung, SSD, high, o.G.

#	TP#2	TP#4	TP#6
1	110,85	43,81	30,53
2	108,12	40,85	30,59
3	113,24	37,14	30,70
4	108,42	41,31	30,37
5	106,20	39,38	30,61
6	113,15	41,31	30,90
7	109,27	44,02	30,50
8	111,79	37,10	30,17
9	108,32	39,43	30,43
10	108,73	40,20	30,31
MW	109,81	40,46	30,51

Tabelle A.97: Testszenario #5: Standardabweichung, HDD, high, o.G.

#	TP#1	TP#3	TP#5
1	408,92	391,69	318,33
2	427,49	410,14	306,65
3	448,61	455,88	305,94
4	412,12	431,92	307,39
5	421,90	404,95	306,17
6	427,42	403,51	306,14
7	392,98	437,92	306,34
8	417,59	445,33	307,10
9	402,22	413,07	305,97
10	432,53	412,18	305,89
MW	419,18	420,66	307,59

Tabelle A.98: Testszenario #5: Standardabweichung, SSD, high, m.G.

#	TP#2	TP#4	TP#6
1	429,82	400,83	318,14
2	421,05	444,14	307,24
3	408,74	421,98	307,48
4	403,05	409,46	315,04
5	411,15	408,65	306,83
6	426,74	425,46	314,93
7	400,20	437,71	305,15
8	431,86	430,23	303,87
9	408,93	416,25	305,34
10	442,68	421,00	307,09
MW	418,42	421,57	309,11

Tabelle A.99: Testszenario #5: Standardabweichung, HDD, high, m.G.

Varianz

#	TP#1	TP#3	TP#5
1	7,75	8,05	7,68
2	7,70	7,69	7,73
3	7,72	7,70	7,72
4	7,69	7,99	7,71
5	7,60	7,72	7,74
6	8,07	7,71	8,10
7	7,70	7,61	7,71
8	7,71	7,72	7,70
9	7,70	7,76	7,70
10	8,12	7,66	7,69
MW	7,78	7,76	7,75

Tabelle A.100: Testszenario #5: Varianz, SSD, low, o.G.

#	TP#2	TP#4	TP#6
1	7,65	7,59	7,60
2	7,73	7,73	7,92
3	7,75	7,67	7,71
4	7,72	7,84	7,76
5	7,72	7,79	7,77
6	7,61	7,81	7,72
7	8,15	7,73	7,59
8	7,73	7,75	7,76
9	7,60	7,69	7,73
10	7,71	8,36	7,76
MW	7,74	7,80	7,73

Tabelle A.101: Testszenario #5: Varianz, HDD, low, o.G.

#	TP#1	TP#3	TP#5
1	87,93	77,58	76,74
2	77,90	75,21	73,72
3	74,67	74,55	73,79
4	76,19	74,68	74,18
5	75,03	73,68	74,47
6	74,83	74,38	74,58
7	74,79	74,43	74,09
8	74,67	74,56	73,87
9	76,13	74,18	73,92
10	75,86	74,13	74,56
MW	76,80	74,74	74,39

Tabelle A.102: Testszenario #5: Varianz, SSD, low, m.G.

#	TP#2	TP#4	TP#6
1	79,55	78,42	78,66
2	73,91	74,02	73,82
3	74,80	74,81	76,56
4	76,22	73,71	74,95
5	74,19	73,81	74,09
6	74,15	74,43	74,52
7	74,94	74,29	74,20
8	74,36	74,50	73,92
9	74,60	76,05	74,61
10	74,99	74,70	74,74
MW	75,17	74,87	75,01

Tabelle A.103: Testszenario #5: Varianz, HDD, low, m.G.

#	TP#1	TP#3	TP#5
1	16,70	15,46	15,29
2	16,46	15,30	15,22
3	15,62	15,36	15,41
4	16,70	15,34	15,25
5	16,18	15,36	15,52
6	16,59	15,44	15,21
7	16,61	15,24	15,49
8	16,41	15,41	15,43
9	16,95	15,28	15,29
10	16,84	15,24	15,33
MW	16,51	15,34	15,34

Tabelle A.104: Testszenario #5: Varianz, SSD, med, o.G.

#	TP#2	TP#4	TP#6
1	17,11	15,30	15,40
2	16,77	15,29	15,45
3	16,74	15,32	15,44
4	18,68	15,32	15,55
5	17,42	15,47	15,29
6	16,97	15,28	15,76
7	17,09	15,19	15,20
8	17,06	15,52	16,00
9	16,44	15,20	15,23
10	16,57	15,34	15,23
MW	17,09	15,32	15,46

Tabelle A.105: Testszenario #5: Varianz, HDD, med, o.G.

#	TP#1	TP#3	TP#5
1	181,07	155,78	150,54
2	187,83	149,02	147,20
3	196,77	147,98	149,09
4	183,59	148,32	148,48
5	198,02	147,39	147,99
6	187,00	148,23	150,88
7	183,96	149,32	148,28
8	175,31	148,21	148,50
9	172,16	148,17	148,70
10	186,33	148,54	148,11
MW	185,20	149,10	148,78

Tabelle A.106: Testszenario #5: Varianz, SSD, med, m.G.

#	TP#2	TP#4	TP#6
1	210,95	150,98	156,09
2	166,36	149,61	149,35
3	159,20	148,43	148,23
4	246,76	148,72	149,15
5	158,41	149,57	147,77
6	184,57	148,81	148,30
7	161,07	148,85	148,00
8	259,37	150,46	149,61
9	210,63	148,92	148,24
10	217,12	147,86	148,01
MW	197,44	149,22	149,28

Tabelle A.107: Testszenario #5: Varianz, HDD, med, m.G.

#	TP#1	TP#3	TP#5
1	36,61	32,31	30,29
2	36,89	33,81	30,97
3	36,78	33,73	30,40
4	37,44	34,06	30,39
5	36,56	34,95	30,43
6	37,00	32,76	30,66
7	35,64	33,95	30,91
8	36,12	33,46	30,39
9	37,31	32,39	30,44
10	35,33	34,40	30,38
MW	36,57	33,58	30,53

Tabelle A.108: Testszenario #5: Varianz, SSD, high, o.G.

#	TP#2	TP#4	TP#6
1	41,10	36,92	30,36
2	39,67	33,60	30,37
3	37,80	33,86	30,97
4	41,91	32,53	30,60
5	41,09	31,77	30,01
6	39,02	34,81	30,15
7	41,52	32,36	30,24
8	39,56	33,89	30,51
9	38,61	33,93	30,17
10	40,78	31,30	30,66
MW	40,11	33,50	30,40

Tabelle A.109: Testszenario #5: Varianz, HDD, high, o.G.

#	TP#1	TP#3	TP#5
1	385,05	384,93	321,63
2	390,96	406,39	310,82
3	415,97	443,64	307,98
4	372,79	348,04	306,56
5	412,70	397,53	306,21
6	370,34	416,35	306,03
7	404,66	421,97	308,02
8	403,24	382,04	306,86
9	414,81	454,25	307,00
10	373,67	392,79	306,44
MW	394,42	404,79	308,76

Tabelle A.110: Testszenario #5: Varianz, SSD, high, m.G.

#	TP#2	TP#4	TP#6
1	426,38	337,83	320,76
2	404,60	339,60	309,66
3	407,63	360,79	307,46
4	393,60	379,72	379,27
5	411,20	379,49	322,17
6	406,39	332,99	355,23
7	420,89	357,14	377,24
8	417,27	319,80	358,37
9	417,89	405,73	316,20
10	396,25	373,67	346,21
MW	410,21	358,68	339,26

Tabelle A.111: Testszenario #5: Varianz, HDD, high, m.G.

Testszenario #6: R und DBI

#	TP#1	TP#3	TP#5
1	33,35	31,17	31,54
2	33,27	31,22	31,48
3	33,56	31,38	31,52
4	33,32	31,21	31,64
5	33,57	31,23	31,54
6	33,43	31,30	31,59
7	33,52	31,26	31,63
8	33,36	31,31	31,60
9	33,36	31,20	31,64
10	33,31	31,35	31,79
MW	33,41	31,26	31,60

Tabelle A.112: Testszenario #6: SSD, low

#	TP#2	TP#4	TP#6
1	35,60	31,14	31,70
2	35,75	31,32	31,74
3	36,00	31,17	31,86
4	36,66	31,47	31,78
5	36,53	31,13	31,89
6	35,96	31,47	31,76
7	36,02	31,21	31,79
8	36,35	31,28	31,75
9	36,13	31,13	31,73
10	36,05	31,29	31,95
MW	36,11	31,26	31,80

Tabelle A.113: Testszenario #6: HDD, low

#	TP#1	TP#3	TP#5
1	232,30	76,92	74,42
2	227,07	76,70	73,76
3	226,22	77,06	74,65
4	224,96	76,70	74,63
5	226,71	76,35	74,53
6	229,89	76,60	74,47
7	226,08	77,57	74,40
8	238,93	77,55	74,48
9	234,04	76,64	74,93
10	229,83	76,32	74,36
MW	229,60	76,84	74,46

Tabelle A.114: Testszenario #6: SSD, med

#	TP#2	TP#4	TP#6
1	255,92	78,98	73,62
2	249,27	78,90	73,39
3	251,48	78,38	73,42
4	240,72	79,26	74,13
5	231,68	79,17	73,69
6	240,78	79,12	74,06
7	237,15	78,89	73,76
8	238,67	78,62	73,96
9	237,67	78,63	73,99
10	237,87	78,71	74,15
MW	242,12	78,87	73,82

Tabelle A.115: Testszenario #6: HDD, med

#	TP#1	TP#3	TP#5
1	17583,53	787,51	180,07
2	17280,61	773,14	179,98
3	17318,93	724,68	180,23
4	17207,37	758,90	180,25
5	17298,50	768,98	179,39
6	17281,36	745,28	179,11
7	17265,94	744,85	180,26
8	17248,35	724,38	180,08
9	17298.92	769,14	179,41
10	17302,86	749,40	180,35
MW	17309,72	754,63	179,91

Tabelle A.116: Testszenario #6: SSD, high

#	TP#2	TP#4	TP#6
1	19744,23	713,57	183,34
2	19563,53	757,65	180,86
3	19521,58	724,87	182,08
4	19582,18	711,25	183,66
5	19586,48	768,92	183,22
6	19537,59	749,41	179,91
7	19598,14	761,67	181,14
8	19508,33	779,44	182,48
9	19581,34	785,68	181,00
10	19544,79	745,78	181,90
MW	19576,82	749,82	181,96

Tabelle A.117: Testszenario #6: HDD, high

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und dabei keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht. Die Arbeit wurde bisher weder gesamt noch in Teilen einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Neuss, 16.08.2019