

Vergleich der Performance von Datenbanksystemen in Django-Webanwendungen

Autor: Nick Reichelt
Praxisbetrieb: Deutsche Telekom IoT GmbH
Betreuer: Prof. Dr. rer. nat. Andreas Thor

Matrikelnummer: 79144
Studiengruppe: 21-TIB
E-Mail: nick.reichelt@stud.htwk-leipzig.de

Problemstellung und Motivation

- Hintergrund**
- Viele bestehende Untersuchungen im Hinblick auf Performance für verschiedene Datenbankmanagementsysteme (DBMS) anhand von Testfällen existieren bereits
 - Keine zusammenhängende Analyse aller DBMS, die von Django genutzt werden bezüglich Effizienz
- Fehlende Aussage über den Performancevergleich
- Problem**
- Ladezeiten bestimmter Seiten der Webanwendung sind überdurchschnittlich hoch (> 2s)
 - Beeinträchtigung der Benutzerfreundlichkeit sowie der Performance der Anwendung
- Motivation**
- Effiziente Webanwendung als unverzichtbares Werkzeug für Geschäftsabläufe & Nutzerzufriedenheit
 - Ineffiziente Anwendungen führen zu: steigender Nutzerunzufriedenheit, erhöhte Nutzungsdauer der Seite für „schnelle“ Aufgaben, Fruststeigerung beim Entwicklungsprozess, erhöhte Kosten

Untersuchungsdesign

- Untersuchungsfrage**
- Inwiefern beeinflusst der reine Austausch eines DBMS die Performance der Webanwendung unter ansonsten gleichbleibenden Bedingungen?
- Ziele des Projektes**
- Ineffiziente Unterseiten (Use Cases) identifizieren, Vergleichskriterien festlegen und IST-Zustand erfassen
 - Entwicklung eines automatisierten Skripts zur Anfragesimulation & Datenerfassung
 - Implementierung von 4 verschiedenen DBMS + Schema- & Datenmigration
 - Identische Datenerfassung + Auswertung & Vergleich anhand der Laufzeitunterschiede der identifizierten Use Cases
- Planung in Arbeitspakete**
- Literatur-recherche

technische Einarbeitung

Festlegung der Aufgaben






Entwicklung Skript

Austausch der DBMS

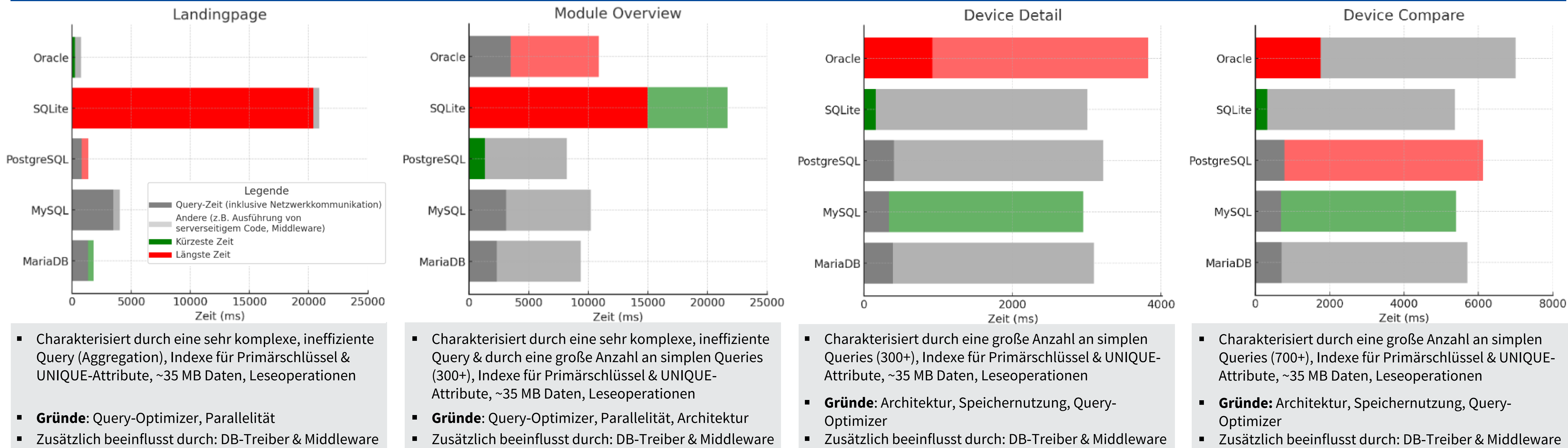
Daten-erfassung

Vergleich der DBMS

Vergleich der verwendeten Datenbankmanagementsysteme

	 Abbildung 1	 Abbildung 2	 Abbildung 3	 Abbildung 4	 Abbildung 5
Architektur	Relationales DBMS mit Client-Server Struktur (v. 23.5.0.24.07)	Serverlose, eingebettete Datenbank in einer einzigen Datei (v. 3.46.1)	Relationales DBMS mit Client-Server Struktur (v. 16.4)	Relationales DBMS mit Client-Server Struktur (v. 8.0.39)	Relationales DBMS mit Client-Server Struktur, Fork von MySQL (v. 11.3)
Query-Optimizer	Kostenbasierter Optimizer, adaptive Query-Optimierung und SQL Plan Management verfügbar; umfangreiche Nutzung von Statistiken	Heuristischer Optimizer mit Index-Auswahl	Fortschrittlicher kostenbasierter Optimizer mit paralleler Abfrageplanung & Index-Auswahl, Unterstützung für Nested-Loop Joins, Merge Joins & Hash Joins, Partitionierung & Partition-Pruning	Kostenbasierter Optimizer mit Index-Auswahl, Unterstützt Nested-Loop Joins, Partitionierung & Partition-Pruning	Ähnlich wie MySQL mit Unterstützung für Nested-Loop Joins, Hash Joins und Block-Nested-Loop Joins, Partitionierung & Partition-Pruning
Parallelität	Parallele SQL-Verarbeitung, parallele Scans, Aggregationen & Joins durch Multi-Threading	Single-Threading	Parallele Sequenz-Scans, Joins und Aggregationen durch Multi-Processing (mit separaten Worker-Prozessen)	Begrenzte Unterstützung für parallele Abfragen, in der Regel Single-Threading	Experimentelle Unterstützung für parallele Joins durch Multi-Threading
Speichernutzung	Dynamische RAM-Nutzung, geringer Festplattenzugriff [1]	8 MB Cache, danach Festplattenspeicher [2]	128 MB Shared Buffer, danach Festplattenspeicher [3]	128 MB InnoDB Buffer, danach Festplattenspeicher [4]	128 MB InnoDB Buffer, danach Festplattenspeicher [5]

Laufzeitvergleich anhand von unterschiedlichen Use Cases



Fazit und Ausblick

- Fazit**
- Oracle ist darauf ausgelegt, auch komplexe Queries performant auszuführen, reagiert aber schwerfällig auf eine große Anzahl an simplen Queries
 - SQLite ist performant gegenüber einfachen Queries, aber weist große Probleme mit komplexeren auf
 - MySQL & MariaDB mit moderater Performance gegenüber allen Testfällen
 - PostgreSQL im Durchschnitt mit der besten Leistung, da es sowohl komplexe als auch viele simple Queries in einer performanten Art und Weise ausführt
- Ausblick auf Bachelorarbeit**
- Grundlage ist PostgreSQL als DBMS für weitere Performanceoptimierungen
 - Untersuchung der Effizienz von Query-Optimierungsansätzen anhand von Antipatterns
 - Untersuchung der Effizienz einer Caching-Strategie

Literatur und Quellen

- Literaturquellen**
- [1] Oracle Corporation. (2024). Oracle Database Documentation 23c. Abgerufen von <https://docs.oracle.com/en/database/oracle/oracle-database/23/>
- [2] Hipp, D. R., & SQLite Development Team. (2024). SQLite Documentation. Abgerufen von <https://sqlite.org/docs.html>
- [3] PostgreSQL Global Development Group. (2024). PostgreSQL 16 Documentation. Abgerufen von <https://www.postgresql.org/docs/16/>
- [4] Oracle Corporation. (2024). MySQL 8.0 Reference Manual. Abgerufen von <https://dev.mysql.com/doc/refman/8.0/en/>
- [5] MariaDB Corporation. (2024). MariaDB Documentation for Version 11.3. Abgerufen von <https://mariadb.com/kb/en/documentation/>
- Bildquellen**
- Abbildung 1: <https://it-dialog.com.ua/en/vendors/oracle-corporation.html>
- Abbildung 2: <https://www.pngegg.com/en/search?q=sqlite>
- Abbildung 3: <https://medium.com/codex/intro-to-postgresql-c8da31335c34>
- Abbildung 4: <https://de.cleanpng.com/png-yeqxoj/>
- Abbildung 5: <https://www.librebyte.net/en/tag/mariadb/>