

Projektarbeit

Internationale Hochschule Duales Studium

Studiengang: Informatik

**Wie beeinflusst die Asynchronität von Operationen innerhalb eines Node.js Express Endpunkts die Performance in Lasttest?**

Müller Korbinian

Matrikelnummer: 102302316

Adresse

Lohwald Straße 59

86356 Neusäß

Abgabedatum: 30.09.2024

Inhalt

[Einleitung 3](#_Toc178023477)

[Theoretische Fundierung 3](#_Toc178023478)

[Begriffserklärung 3](#_Toc178023479)

[Node.JS und Express 3](#_Toc178023480)

[Asynchrone Programmierung 4](#_Toc178023481)

[Lasttest und Performance-Metriken 4](#_Toc178023482)

[Theoretische Grundlagen 5](#_Toc178023483)

[Zusammenhang zwischen Asynchronität und Performance 5](#_Toc178023484)

[Bedeutung für die serverseitige Anwendung 5](#_Toc178023485)

[Methodik 5](#_Toc178023486)

[Testumgebung 5](#_Toc178023487)

[Datenbank und Datenquellen 6](#_Toc178023488)

[Aufbau der Lasttests 7](#_Toc178023489)

[Implementation der Endpunkte 8](#_Toc178023490)

[Ergebnisse 8](#_Toc178023491)

[Testergebniss 8](#_Toc178023492)

[Performance Vergleich 8](#_Toc178023493)

[Fazit 8](#_Toc178023494)

[Literaturverzeichnis 8](#_Toc178023495)

Einleitung

In der Heutigen Zeit steht die Effizienz und Performance an erster Stelle bei Software. Insbesondere wenn es sich dabei um eine Webanwendung handelt. Das Thema wird besonders Interessant wenn diese in der Lage sein sollte eine sehr große Nutzerzahl standzuhalten. Eine Technologie die Entwickelt wurde, um mit den immer mehr Ansteigenden Anforderungen zurecht zukommen ist Node.js. Dies ist ein Platform die es ermöglicht JavaScript auf dem Server auszuführen. Dies wird durch die V8 Engine von Google ermöglicht. Ein Alleistellungsmerkmal von Node.js ist die Möglichkeit Code Asynchron, sprich Nicht blockierend, auszuführen.

Diese Arbeit untersucht, wie sich Asynchronität von Operationen innerhalb eines Node.js Express-Endpunkt sich auf die Performance auswirkt. Es wird Untersucht wie und in welchem Maße die asynchrone Verabeitung an Anfragen zu einer Höheren Effizienz und einer geringeren Antwortzeit führt. Dazu wird ein Experiment durchgeführt, bei dem zwei Express.js Endpunkte implementiert werden. Einer verarbeitet die Anfragen Asynchron und der andere Synchron. Beide Endpunkte werden einem Lasttest unterzogen und die Ergebnisse werden miteinander verglichen.

Das Ziel dieser Arbeit ist es, die Auswirkung der asynchronen Programmierung auf den Webserver zu quantifizieren. Dies wird durch ein praxisnahes Experiment realisiert. Die Ergebnisse helfen eine Fundierte Aussage über Vorteile und Grenzen zu treffen.

Theoretische Fundierung

Dieser Abschnitt der Arbeit beschäftigt sich mit den Theoretischen Grundlagen. Die Weiteren abschnitte basieren auf dem hier dargelegten Wissen. Zunächst werden Grundlegende Begriffe erläutert. In diesem Abschnitt wird erklärt, was unter Node.JS, Express, Asynchrone Programmierung und Lastentest zu verstehen sind. Im darauffolgen Abschnitt werden die Theoretischen Grundlagen erläutert auf denen die Tests basieren.

Begriffserklärung

In dieser Ausführung werden technische Konzepte und Technologien erwähnt. Damit jeder Leser auch die späteren Technisch Anspruchsvollen Passagen versteht werden in den Folgenden Abschnitte die wichtigsten Grundbegriffe ausführlich erläutert.

Node.JS und Express

Node.JS ist ein JavaScript Framework, dass die durch Google Entwickelte V8 Engine, nutzt, um JavaScript außerhalb des Webbrowsers auszuführen. Dabei wird der Quellcode Asynchron ausgeführt. Beim Start des Node.JS Prozesses wird eine Eventloop gestartet, die auf eingehende Events reagiert und diese Asynchron abarbeitet (Huang and Cai, 2018, S.1-3).

Anhand dieser einzigartigen Architektur ist Node.JS in der Lage sehr effizient zu Arbeiten. Mit einer Speichergröße von 8 GB sind maximal 40000 Verbindungen mit dem Webserver möglich. Im Vergleich dazu sind bei herkömmlichen Serverseitigen Programmiersprachen wie Java oder PHP, in etwa 4000 Nutzer möglich. Dies beruht auf der Annahme, dass die Herkömmlichen Programmiersprachen in etwa 2MB Arbeitsspeicher für jeden neuen Thread benötigen. Neben diesen Stärken weißt Node auch eine Schwäche für CPU intensive Aufgaben auf(Huang, 2020).

Laut der Entwicklerumfrage von Stackoverflow (<https://survey.stackoverflow.co/2024>) ist NodeJS eine der Begehrtesten Programmiersprachen. Kein Wunder dass auch auf Node.JS selber bei den Web-Frameworks an Zweiterstelle ist. An siebter Stelle ist das Framework Express.JS, dieses basiert auf Node.JS. Anhand dieser Popularität wurde das Framework Express.JS für diese Arbeit ausgewählt. Die Entwickler von Express beschreiben dieses als ein schnelles und minimalistisches Web-Framework für Node.js. Es bietet eine große Anzahl an HTTPS Funktionalitäten, dadurch wird es erleichtert schnell und leicht, robuste APIs zu entwickeln (<https://expressjs.com>).

Asynchrone Programmierung

Unter Asynchronität in der Programmierung versteht man, wenn der Programmcode nicht Kontinuierlich ausgeführt wird. Dieser wartet bis Daten angekommen sind oder ein Event ausgelöst wird. Dadurch werden die Systemresourcen freigegeben, wenn diese nicht benötigt werden. Diese Funktionalität ist sehr stark in JavaScript integriert (Flanagen, 2020 S.342).

Es ist wichtig zu erwähnen, dass Java und Node.JS nutzen bei der Ausführung von Quelltext nur einen Thread. Durch die Asynchrone Auslegung von Node.JS ist es dennoch möglich einen hohen Grad an parllität in der Ausführung zu erreichen. Es kommt zu einer Performance, die in anderen Programmiersprachen mittels Threads erreicht wird, mit einem Bruchteil der Resourcennutzung (Flanagen, 2020 S.583).

Lasttest und Performance-Metriken

Eines der wichtigsten Themen das beim Testen anfällt ist der Test der Performance. Dabei muss nicht die gesamte Software Umgesetzt sein, sobald die ersten Elemente umgesetzt sind kann damit angefangen werden. Im Bereich der Webentwicklung ist eine der Wichtigsten Fragen, welches die maximale Anzahl an Nutzern ist. Damit die Ergebnisse aussagekräftig sind, ist es essenziell das die Test so nah an der produktiv Umgebung ist wie möglich (Kleuker, 2019 S.355-356).

In seinem Buch beschreibt Kleuker(2019), dass eine weitere Maßnahme für die Qualitätsbewertung sind Metriken. Dabei wird ein Kriterienkatalog mit relevanten Aspekten erstellt. Im Bereich Softwareentwicklung können die Aspekte automatisiert überprüft und ausgewertet werden (Kleuker, 2019 S22-23).

Für die Evaluation in dieser Projektarbeit werden folgende Metriken ausgewertet:

* Antwortzeit: Die benötigte Zeit die Anfrage an den Server zu schicken, diese zu Bearbeiten und das Ergebnis zurück zu Senden. Die Zeit wird in Millisekunden, MS, gemessen
* Durchsatz: Die Anzahl an Reuests die in einer Sekunde verarbeitet werden können
* CPU- und Speicherverbrauch: Diese bestimmen wie effizient der Server seine Ressourcen währen der Nutzung einsetzt.

Theoretische Grundlagen

Nach der Erläuterung der grundlegenden Begriffe werden nun die Theoretischen Grundlagen geschildert. Diese dienen als Grundlage für das durchgeführte Experiment. Diese Grundlagen sind entscheidend um die Leistungsmessungen und Lasttests in Bezug zu Asynchrone API Endpunkte im weiteren Verlauf zu verstehen.

Zusammenhang zwischen Asynchronität und Performance

Huang und Cai (2018) haben bereits erläutert das Node.js Hardware nahen Code Asynchron ausführt (Huang and Cai, 2018, S.1-3). Dadurch wird der Zugriff auf die Ressourcen b

Bedeutung für die serverseitige Anwendung

Dadurch entstehen

Methodik

Dieser Abschnitt der Arbeit befasst sich mit dem praktischen Vorgehen. Zunächst wird erläutert wie die Tests strukturiert sind. Im Anschluss wird dargestellt wie sich die Testumgebung zusammensetzt. Darauf folgt die Implementierung der API-Endpunkte.

Testumgebung

Um die Experimente durchzuführen wird ein Webserver mit der Node.js Framework Express.js entwickelt. Mittels dem Framework Express.js wird ein REST-API Service implementiert. Die API stellt zwei Endpunkte zur verfügung, einen Synchronen und einen Asynchronen. Beide Endpunkte verarbeiten eingehende Anfragen auf die gleiche Weise. Nach dem Empfang des Request wird auf die MySQL Datenbank zugegriffen und ein SQL-Statement ausgeführt. Das Ergebnis der Datenbank abfrage wird in Form eines JSON-Objekt zurückgegeben.

Der Express.JS Server wird direkt auf dem PC Ausgeführt. Hierbei wird der Port 3000 benutzt. Die Datenbank wird einem Docker Container gestartet. Der Datenbank Container wird mittels Docker Compose gestartet. Hier wird die Version 3.9 des Docker Compose File benutzt. Das MySQL Image nutzt die Version 8.0. Die Datenbank ist auf dem lokalen Netzwerk unter dem Port 3306 Erreichbar.

Die Hardware des Desktop-PCs, auf dem die Tests durchgeführt werden, besteht aus:

* Prozessor: IntelCore I9-9900 KF, nicht übertaktet
* Arbeitsspeicher: 32 GB DDR-4, 2100 GHZ
* Grafikkarte: Nvidia RTX 2080 Super

Folgendes sind die Software die Für die Entwicklung und Durchführung des Experimentes Genutzt worden sind:

* Betriebssystem: Windows 10 Professional
* IDE: WebStorm, JetBrains
* Docker: Version XX, 16 GB Arbeitsspeicher
* Node.js: Version 20.11.1
* Express.js: Version 10.2.4
* Artillery, Version , Software für Lasttest.

Datenbank und Datenquellen

Die Daten, die für den Lasttest genutz werden, stammt von der Platform Kaggle. Die Daten sind unter [Laptop Prices (kaggle.com)](https://www.kaggle.com/datasets/owm4096/laptop-prices) zu erreichen. Dieser Datensatz enthält folgende Informationen zu Laptops: Hersteller, Produktname, Technische Spezifikationen, Bildschirmgröße und Preis. Die Daten stehen in Form einer CSV Datei öffentlich zum Download bereit.

**Datenbankaufbau**

Für die Verbindung zu der Datenbank wurde die Administrationsoberfläche von HeidiSQL benutzt. In dieser wurde eine Neue Datenbank mit dem Namen Sapphire angelget. In dieser Datenbank wurde eine Tabelle mit dem Namen laptop\_prices angelegt. In dieser wurde mittels der Funktionalität, Import CSV, von HeidiSQL die Daten Importiert. Die wichtigsten Felder in der neu Angelegten Datenbank sind:

* Company: Der Hersteller des Laptops.
* Product: Das spezifische Laptop-Modell.
* TypeName: Der Typ des Laptops (z.B. Ultrabook, Notebook).
* Inches: Bildschirmgröße in Zoll.
* Ram: Arbeitsspeicher in GB.
* OS: Das Betriebssystem.
* Weight: Gewicht des Laptops in kg.
* Price\_euros: Der Preis des Laptops in Euro.

Die Datenbank ist so strukturiert, dass alle relevanten Informationen schnell und effizient abgerufen werden können.

In der Ausgeführten Abfrage werden alle Relevanten Daten zu den Laptops zusammengetragen, insbesondere die Komplete System Speichergröße. Alle Elemente werden Nach den folgenden Kriterien gefiltert:

* Preis liegt zwischen 500€ und 2000€
* Die Taktfrequenz der CPU muss über 2.5 GHz liegen
* Die Displaygröße liegt zwischen 13 und 17 Zoll
* Der Laptop besitzt mehr als 8 GB Arbeitspeicher

Die Rechner werden Nach der System Speicherkapazität sortiert und die ersten 10 Elemente zurückgegeben.

Aufbau der Lasttests

Der Aufbau der Lasttest ist ausschlaggebend um realistische und aussagekräftige Resultate zu erreichen. In diesem Abschnitt wird erläutert, wie der Test aufgebaut und durchgeführt wird. Des Weiteren wird erklärt welche Tools zum Einsatz kamen.

**Simuliertes Lastszenario**

Das Lastzenario wird in mehreren Schritten aufgebaut. Die Anzahl der gleichzeitigen Benutzer wird insgesamt in vier Phasen erhöht. In der ersten Phase sind es 100 konkurrente simulierte Nutzer. In Phase 2 werden 250 Nutzer simuliert. In der dritten sind es 500. In der Letzten Phase wird die Funktionalität „Ramp up“ von Artillery genutzt. Dabei wird die Anzahl an Nutzer Langsam erhöht. Das Ziel ist es bis maximal 1000 Simulierte Nutzer zu erlangen. Dadurch wird eine realistische Beanspruchung Simuliert.

Insgesamt wird dieser Testablauf für jeden Endpunkt 3 mal wiederholt.

**Antwortzeit und Durchsatz**

Diese beiden Punkte sind die wichtigsten in der Leistungsbeurteilung einer REST-API. Die Antwortzeit beschreibt die Dauer, die der Server benötigt um die Anfrage zu verarbeiten. Für ein aussagekräftiges Ergebniss zu erreichen berechnet Artillery sowohl die Durchschnittliche Antwortzeit, als auch das 95.Percentil. Dadurch wird sichergestellt die Statistische Anomalität relativiert werden. Ebenso wird die Maximale Antwortzeit für jede der Phasen dokumentiert.

Den Maximalen Durchsatz beschreibt die Anzahl an Anfragen die in einer Zeitspanne von einer Sekunde bearbeitet werden kann. Dies wird in der Letzen Phase ermittelt. In der Artillery Konfiguration wird festgelegt, das die Maximale Antwortzeit nicht mehr als 5000 MS betrage darf.

Insgesamt ist zu erwarten, dass beide Endpunkte eine ähnliche Performace besitzen, wenn der Durchsatz niedrig ist. Dies liegt daran, dass Node.js bereits asynchron Arbeitet. Erst bei Hohem Durchsatz wird erwartet, dass der Asynchrone Endpunkt ein bessere Antwortzeit aufweist.

**Hardware Auslastung**

Im Testbetrieb werden die Ressourcen des Rechners Kontinuierlich überprüft. Dabei wird die CPU und RAM Nutzung des Express.js Servers überprüft und in einer CSV-Logdatei gespeichert. Um diese Daten zu erheben wird ein Batch-Script erstellt und benutzt. Dieses Filtert nach dem Express.js PID. Dadurch wird sichergestellt, dass die Daten nicht durch andere laufenden Prozesse verfälscht werden

**CPU-Auslastung**

Die CPU-Auslastung ist ein wesentliches Merkmal für die Effizienz der Anfrageverarbeitung. Da Node.js nur auf einem Thread ausgeführt wird steigt die CPU-Nutzung mit der Intensivität der Server Beanspruchung. Hier wird erwartet, dass der Asynchrone Endpunkt weniger Ressourcen benötigt wie der Synchrone. Dies basiert auf der Annahme, dass durch die Asynchrone Verarbeitung weniger Ressourcen blockiert werden.

**Arbeitsspeicher-Auslastung**

Die RAM-Auslastung wurde ebenfalls gemessen, da ein hoher Speicherverbrauch auf ineffiziente Speicherverwaltung hindeuten könnte. Vor allem bei komplexen Datenbankabfragen und hoher Benutzerlast sind Schwankungen in der Speicherauslastung zu erwarten.

Implementation der Endpunkte

Die Implementation der Endpunkte erfolgt auf zwei Verschiedenen Paradigmen. Einer wird Synchron und der andere Asynchron entwickelt. Die beiden Implementation unterscheiden sich an zwei wichtigen Stellen. Die Callback Funktion die Aufgerufen wird, wenn eine Anfrage an den Asynchronen Enpunkt gestellt wird ist mit dem Schlüsselwort async versehen. Dadurch wird diese Funktion aufgerufen und Node.js wartet nicht bis die Funktion vollends ausgeführt wurde. In der Zwischenzeit können weitere Anfragen angenommen werden. Der zweite unterschied liegt daran, dass in der Funktion das Schlüsselwort await nutzt. Dies geschieht bei der Datenbankabfrage. Dies veranlasst Node.js die Verarbeitung der Anfrage zu pausieren, bis die Datenbank die entsprechenden Daten liefert.

Durch diese beiden Unterschiede wird der Thread nicht blockiert und Node.js kann mehrere Anfragen quasi Parallel bearbeiten. Dieser Unterschied bietet eine Ideale Grundlage um die Auswirkung von Asynchroner Programmierung auf die Performance einer Rest API zu testen.

Ergebnisse

In diesem Abschnitt werden die Resultate der durchgeführten Tests detailliert dargelegt. Das Ziel ist es, die Leistung der Endpunkte zu quantifizieren. Durch die Ergebnisse können Schlüsse auf die Effizienz und Skalierbarkeit der Implementationen gezogen werden.

Testergebniss

Performance Vergleich

Fazit

Literaturverzeichnis