Ostfalia - Hochschule für angewandte Wissenschaften
Fakultät Informatik
Studiengang Informatik

Master-Arbeit Vorvertrag

Reactive Programming mit Quarkus

eingereicht bei	Prof. Dr. B. Müller	
von	Erik Simonsen	70455429

Wolfenbüttel, den 15. März 2021

1 Problemstellung

In den letzten Jahren haben sich die Anforderungen an große, über das Internet zugängliche Softwaresysteme (Web-Anwendungen) stark verändert. Während damals Antwortzeiten im Sekunden Bereich, regelmäßige Downtime durch Wartungsarbeiten, Datenmengen im Gigabytebereich und eine begrenzte Menge an Servern gängig waren, werden heutzutage hoch skalierbare Cloud-Infrastrukturen genutzt, Antwortzeiten im Millisekunden Bereich und dauerhafte Verfügbarkeit werden erwartet und es sammeln sich Datenmengen im Petabyte Bereich an. Um diese Anforderungen zu erfüllen haben sich in den letzten Jahren nach und nach Eigenschaften herauskristallisiert, die ein System erfüllen muss. Diese wurden unter dem Oberbegriff Reactive Systems zusammengefasst. Um Reactive Systems zu implementieren hat sich das Programmierparadigma Reactive Programming durchgesetzt, dieses verspricht eine bessere Performance bei geringerem Speicherbedarf, indem blockierende Aufrufe vermieden werden. Blockierende Aufrufe führen im Betriebssystem zu Prozess- und damit Kontextwechseln. Diese Wechsel haben einen verhältnismäßig hohen CPU- und Speicher-Overhead. Als Webapplikationen nur mehrere tausend Anfragen pro Minute bearbeiten musste, war dieses, durch imperative Programmierung bedingte, Performance-Bottleneck kaum spürbar. Durch stetig steigende Benutzerzahlen, und Trends in der Webentwicklung wie SPAs wächst die Häufigkeit der Client-Server Kommunikation allerdings so stark, das heutige Enterprise-Anwendungen deutlich mehr Anfragen pro Minute verarbeiten müssen.

In dieser Arbeit wird die Nutzung von Reactive Programming speziell in Java und dessen Ökosystem untersucht. Dabei wird eine imperative (blockierend), und eine reaktive serverseitige Webanwendung implementiert. Anschließend werden die beiden Anwendungen sowohl beim Anfragen statischer Daten, als auch dynamischer Daten (mit Datenbankanbindung) anhand von verschiedenen Benchmarks wie u.A Durchsatz (Req/Sec), Durchsatz in MB, Speicherauslastung miteinander verglichen, sowie der Programmablauf in beiden Anwendungen dargestellt und erläutert.

Für die Implementierung wird das moderne Framework Quarkus genutzt, welches wiederum auf dem ereignisorientierten Framework Vert.x aufbaut. Darüber hinaus werden die Vor- und Nachteile des Programmierparadigmas und die Eigenschaften von Reaktiven Systemen, sowie Alternativen und Umsetzungen in anderen Programmiersprachen erläutert.

2 Einzelaufgaben

- Überblick und besseres Verständnis der Größe des Ökosystems (welche Libraries sind relevant), und Abgrenzung der Infrastruktur auf die Quarkus aufbaut (Vert.x (sowie dessen APIs wie Mutiny), Netty)
- Imperative, blockierende API erstellen
- Reaktive, asynchrone API erstellen
- Client schreiben, welcher die APIs den Belastungstests unterzieht

- Ermitteln welche Metriken relevant sind, und wie diese gemessen und grafisch ausgewertet werden können
- Reproduzierbare Ergebnisse ermöglichen (Ablauf in Docker-Container mit fest definierten Parametern)
- Ermitteln welche Faktoren die Performance zusätzlich beeinflussen (Warm-Up von JIT-Compiler, Netzwerk usw.)

3 Angestrebte Ergebnisse

- Anschauliche Erläuterung von Prozessen & Threads, Dispatching und Thread-Abstraktion in Java
- Umfassende Erklärung von Reactive Programming (Abgrenzung zu anderen Paradigmen), Asynchronität und dem Java Tooling / Ökosystem
- Detaillierten, anschaulichen Systemablauf sowohl bei blockierendem System, als auch bei nicht-blockierendem System
- Zeigen, dass reaktive Systeme bessere Ergebnisse in den messbaren Performance-Metriken liefern bzw. unter welchen Bedingungen sie das nicht tun
- Motivation & Anwendungsgebiete für reaktive Systeme erläutern (Vorteile und Nachteile darstellen, reaktive Programmierung ist kein Allheilmittel)
- Sinnvolle Alternativen darstellen und Ausblick auf die Zukunft

4 Ablauf

- Literaturrecherche (Wissenschaftlichen Wert der Literatur beurteilen)
- Einzelaufgaben bearbeiten
- Implementierung der notwendigen Software
- Tests/Messungen durchführen
- Analysieren & Veranschaulichen der Messergebnisse
- Verfassen der Arbeit
- Korrekturarbeiten

5 Notwendige Voraussetzungen

-

6 Meilensteinplan

Bezeichnung	Plan-Termin	Adaptierter Plan-Termin	Ist-Termin
Anmeldung der Arbeit	05.04.2021[1]	-	-
Literaturrecherche & Ökosystem verstehen	12.04.2021	-	-
Implementierung der notwendigen Software (Client & APIs)	03.05.2021	-	
Ermitteln der relevanten Messmetriken & deren Implementierung, sowie Reproduzierbarkeit mit sinnvollen Randbedingungen sicherstellen	17.05.2021	-	-
Durchführen der Tests/Messungen	24.05.2021	-	-
Analysieren & Veranschaulichung der Messergebnisse (Plotting Skript)	31.05.2021	-	-
Verfassen der Arbeit	31.08.2021	-	-
Korrekturarbeiten und Drucken & Binden	13.09.2021	-	-
Abgabe	17.09.2021	20.09.2021	

[1]24 KW Dauer -> 05.04.2021 - 20.09.2021

7 Vorläufige Gliederung

1. Einleitung
   (a) Motivation
   (b) Ziele der Arbeit
   (c) Aufbau
2. Grundlagen
   (a) Reaktive Programmierung
      i. Kontextabgrenzung
      ii. Java-Threads / Prozesswechsel
      iii. Vorteile & Nachteile
      iv. Alternativen
   (b) Reaktive Systeme
      i. Eigenschaften
      ii. Anwendungsgebiete
   (c) Werkzeuge / Tooling
      i. Java Ökosystem

   ii. Andere

3. Vergleich Reaktives & Imperatives System
   (a) Implementierung
   (b) Testbedingungen
   (c) Vorgehen
   (d) Test - Statische Daten
      i. Systemablauf
      ii. Resultate
      iii. Auswertung
   (e) Test - Datenbankzugriffe
      i. Systemablauf
      ii. Resultate
      iii. Auswertung
   (f) Fazit
4. Zusammenfassung