Verteilte Systeme
Praktikum

Ronald Moore, Michael von Rüden, Stephan Gimbel

Hochschule Darmstadt
Fachbereich Informatik

Wintersemester 2023/24

1 Übersicht

Im Rahmen des Praktikums Verteilte Systeme soll eine fiktive Anwendung aus dem Bereich des *Robotik- bzw. Roboterfussball* entwickelt werden.[1] Dazu sollen die Technologien *Sockets*, *Remote Procedure Calls (RPCs)* sowie *Message-Oriented-Middleware (MOM)* verwendet werden. Die Anwendung entsteht dabei Schritt für Schritt und jede der unten stehenden Aufgaben erweitert das System um eine Komponente oder Funktionalität.

In mehreren Phasen wird jeweils ein Teil des in Abbildung 1 dargestellten Gesamtsystems erstellt.

Am Ende müssen mehrere Roboter, die über einen Controller gesteuert werden, mittels eines verteilten Leader-Election Algorithmus einen Kapitän wählen. Das Gesamtsystem soll möglichst robust sein und Aufälle einzelner Systeme, z.B. eines Roboters, verkraften können.

Beachten Sie: Der Fokus liegt auf der Kommunikation der verteilten Komponenten untereinander sowie dem Software-Design, dem Testen und dem Ausrollen (Deployen) der verteilten Anwendungen.

[1] Warnung: Es werden weder echte Robotern verwendet werden, noch eine fachlich korrekten Einführung in der Robotik gegeben. Es geht hier um verteilten Systemen, nicht um "richtige"Robotik.

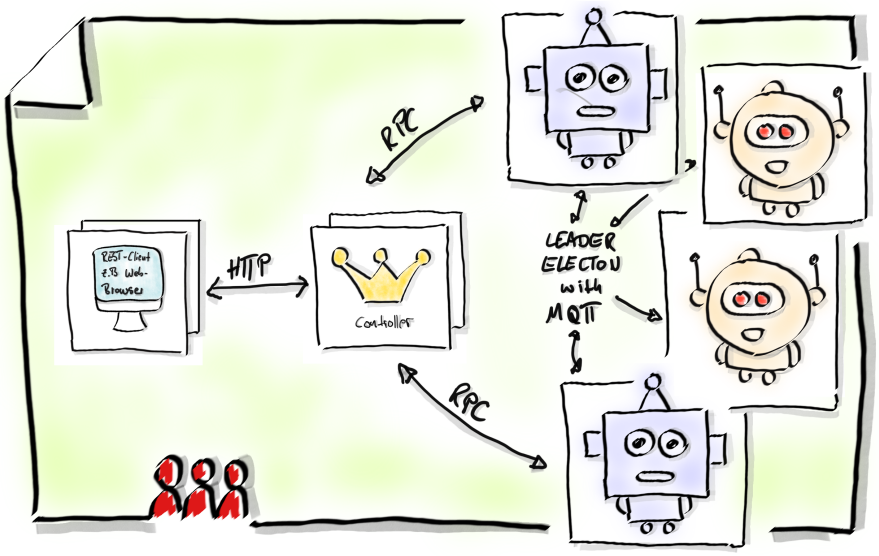

Abb. 1. Das Gesamtsystem mit einem Client, einem hoch-verfügbaren Controller und mehreren Robotern.

2 Grundregeln

Folgende Regeln gelten für das gesamten Praktikum und für alle Aufgaben:

- Das Bearbeiten der Praktikumsaufgaben findet soweit wie möglich in Zweier-Teams statt. Sie bekommen zu Beginn der Lehrveranstaltung einen Parter zugeteilt.
- Die Praktikumsaufgaben müssen zuhause vor- und nachbereitet werden, d.h. außerhalb der Praktikumstermine.[2]
- Alle Aufgaben müssen spätestens zum letzten Gruppentermin abgenommen sein. Andernfalls gilt die Prüfungsvorleistung (PVL) als *nicht bestanden* und eine Zulassung zur Klausur im folgenden Prüfungszeitraum ist nicht möglich. Jede Gruppe ist für die Abnahme der Testate selbst verantwortlich.
- Jeder Gruppe wird ein Repository im GitLab der H-DA (https://code.fbi.h-da.de) zur Verfügung gestellt. Dieses Repository ist für das Praktikum zu verwenden.
- Die Lösungen müssen im Main-Branch Ihres Repositorys zur Verfügung gestellt werden. Eigene Repositories sind nicht zulässig.
- Es ist ein Build Tool (Make, cMake, Maven, Gradle, etc.) zu verwenden.

[2] Ausnahme: Vor Aufgabe 0 muss nichts vorbereitet werden – außer vielleicht die Aufgabenstellung lesen.

- Die Lösungen müssen containerisiert und mittels Docker und Docker-Compose zu testen sein. Nach Möglichkeit soll die Anwendung im Pi-Lab Cluster laufen.
- Die Aufgaben werden im Rahmen der Praktikumstermine mit dem Dozenten durchgesprochen. Hierbei müssen alle Teammitglieder anwesend sein und die Lösung erklären können.
- Es sind keine Programmiersprachen vorgegeben. Eine Kombination mehrerer Sprachen ist erlaubt.
- Sockets und HTTP müssen nativ, d.h. ohne externe Bibliotheken, implementiert werden. Alle weiteren Komponenten und Bibliotheken, z.B für Thrift, gRPC, MQTT, aber auch für Logging oder die Konfiguration, dürfen frei gewählt werden. Bei Fragen besprechen Sie Ihre Pläne mit Ihrem Dozenten.
- Jede Lösung muss getestet werden. Schreiben Sie zu jeder ihrer Lösungen mindestens einen funktionalen Test sowie mindestens einen nicht funktionalen Test.

3 Bonusregelung

Mit Hilfe des Praktikums kann ein **0.3-Noten-Bonus** für die Klausur erworben werden. Der Bonus kann dann erteilt werden, wenn Sie eine herausragende Gesamtlösung (Aufgaben 0-4) präsentieren, die deutlich über den Anforderungen und den Erwartungen liegt.

Der Bonus ist nur einmal gültig – nämlich exakt für die in diesem Semester anstehende Klausur. Der Bonus wird zudem nur dann wirksam, wenn Sie die Klausur auch ohne Bonus mit mindestens der Note 4.0 bestehen.

4 Aufgabenstellung

4.1 Aufgabe 0 Vorbereitung

Der erste Praktikumstermin dient der Vorbereitung und Planung des gesamten weiteren Praktikums. Finden Sie sich in Ihrem Team zusammen. Klären Sie unbedingt gegenseitige Erwartungshaltungen. Bereiten Sie Ihre System (Entwicklungsumgebung, Docker, Kommunikationskanäle, etc.) vor. Testen Sie Ihren GitLab-Zugang und den Zugang zum zur Verfügung gestellten Repository.

In der Vergangenheit hat es sich als überaus nützlich erwiesen, wenn die Teams einen Projektplan und Zeitrahmen erstellt haben. Überlegen Sie gemeinsam, wann wer welche Teile der Softwareentwicklung abgeschlossen haben soll. Nutzen Sie z.B. GitLab Issues und GitLab Milestones für das Projektmanagement.

4.2 Aufgabe 1 - TCP Sockets, HTTP und REST

Im ersten Schritt soll die erste Version des Controllers implementiert werden, der eine REST-Schnittstelle über HTTP anbietet. Über die REST-Schnittstelle sollen zunächst Dummy-Daten mittels HTTP POST entgegengenommen und in einer internen Datenstruktur abgespeichert werden. Zudem bietet der Controller Schnittstellen um vorhandenen Daten mittels HTTP GET abzurufen. Diese Schnittstelle wird im Lauf des Semesters ausgebaut und erweitert.

Es sollen Schnittstellen für mindestens die folgenden Funktionalitäten implementiert werden:

- Abfragen des Status des Gesamtsystems (z.B. vie viele Roboter sind aktiv)
- Abfragen des aktuellen Kapitäns
- Abfragen des Zustands des Controllers (z.B. Health-Check)
- Anstoßen einer (neuen) Kapitänswahl

Der *HTTP* Server des Controllers soll nur mit Hilfe von Sockets und ohne weitere Hilfsmittel, d.h. ohne HTTP-Bibliotheken, implementiert werden.[3]

Der Server soll *HTTP POST* und *HTTP GET* Anfragen akzeptieren. Andere Anfragen sollen abgefangen und mit einer sinnvollen Fehlerbehandlung (Fehlermeldung) behandelt werden. Es ist hierbei erforderlich, dass eingehende HTTP Anfragen komplett und korrekt eingelesen und verarbeitet werden. Das bedeutet u.a., dass GET und POST unterschieden werden müssen und, dass es nicht ausreichend ist, nur die erste Zeile eine HTTP Nachricht zu lesen.

Der Controller ist also ein HTTP-Server. Um den Controller zu testen, brauchen Sie einen HTTP-Client. Sie können als HTTP-Client jeden Browser verwenden, oder frei verfügbare Programme wie ‘curl‘ oder ‘wget‘ verwenden. Sie dürfen auch Ihren eigenen Client programmieren, Ihre Lösung muss aber auch mit jedem Browser oder frei verfügbaren Programmen funktionieren.

Testen Sie Ihr System. Sie sollen einerseits sicherstellen, dass das System *korrekt* funktioniert. Darüber hinaus sollen Sie die *Leistung (Performanz)* Ihres Systems messen. Dafür müssen Sie entscheiden, was genau zu messen ist. Zum Beispiel kann *Latenz* bzw. *Round Trip Time* (RTT) eines HTTP POST gemessen werden. *Maximum Load* bzw. *Transactions per Second* wäre auch angemessen.

Bedenken Sie, dass diese Tests nach jeder späteren Aufgabe wiederholt werden müssen, um zu sehen, wie die Messungen sich ändern wenn neue *Features* dazu kommen!

[3] Externe Bibliotheken dürfen schon für andere Zwecke verwendet werden, z.B. um JSON zu enkodieren bzw. dekodieren, oder Log-Dateien zu schreiben, usw. usf. Sie sollen jedoch HTTP einmal „händisch“ selbst implementieren, um zu sehen, wie einfach es ist, ein Kommunikationsprotokoll zu realisieren und, um Erfahrung mit TCP-Sockets zu bekommen.

Wenn Sie unsicher sind, ob eine Bibliothek erlaubt ist, fragen Sie Ihren Dozenten. Besser noch, stellen Sie die Frage im Moodle-Forum, sodass andere die Antwort auch sehen können.

Werten Sie den Test statistisch valide aus und erstellen Sie ein Messprotokoll, das jeden Test und dessen Ergebnisse darstellt.

4.3 Aufgabe 2 - Remote Procedure Calls (RPC)

Im nächsten Schritt soll das System um eine beliebige Anzahl an Robotern erweitert werden. Die Roboter-Software wird vom Controller mittels eines RPCs, z.B. Apache Thrift oder gRPC, gesteuert. Dazu sollen sich die Roboter beim Start zunächst beim Controller anmelden. Weiterhin braucht es einen Mechanismus um den Zustand (z.B. Health-Check) eines jeden Roboters zu erfassen.

Die RPC Schnittstelle wird in der jeweiligen *Interface Description Language* (IDL) festgehalten. Wenn Sie z.B. Thrift verwenden, fangen Sie diese Aufgabe an, in dem Sie eine Thrift-Datei schreiben.

Die IDL wird auch dafür verwendet, Datenformate für die Datenübertragung zu definieren. Sie sollen nicht selbst die Daten enkodieren bzw. dekodieren.

Es ist wichtig, dass die RPC Schnittstelle sinnvolle Funktionen enthält. Es ist zum Beispiel nicht erlaubt, dass in der IDL-Datei eine Funktion steht, mit einem Text-String als Eingabe und/oder einem Text-String als Ausgabe, wenn die Strings als JSON gedacht sind.[4]

Definieren Sie hier neue Tests, die die Korrektheit der neuen RPC-Schnittstelle feststellen. Definieren Sie auch, wie die Performanz der Schnittstelle gemessen werden kann.

Wiederholen Sie die Performance-Tests aus Aufgabe 1 und vergleichen Sie die Testergebnisse, um festzustellen, ob die Performance der alten Schnittstellen gelitten hat.

Dokumentieren Sie die *alle* Erkenntnisse in einem neuen Messprotokoll und geben Sie es im Moodle ab.

4.4 Aufgabe 3 - Message-oriented Middleware (MoM)

Die Wahl eines (neuen) Kapitäns sollen die Roboter untereinander mit einem verteilten Leader Election Algorithmus organisieren. Hierzu können Sie z.B. einen Ring, Bully oder Raft-Algorithms implementieren. Zum Austausch der benötigten Nachrichten soll eine Message-oriented Middleware, wie z.B. MQTT oder AMQP zum Einsatz kommen.

Definieren Sie dafür die notwendigen *Topics* sowie die Nachrichten, die für den Leader Election Algorithmus erforderlich sind. Überlegen Sie auch, wie Sie eine effiziente Fehlerbehandlung implementieren können, um z.B. den Ausfall eines Roboters zu überleben. Außerdem muss sichergestellt sein, dass die weiteren Voraussetzungen zur Anwendung des Algorithmus eingehalten werden. Sobald

[4] Sie dürfen und sollen Text-Parameter nur für einfache Namen verwenden, nur nicht für Zahlen oder Datenstrukturen. Wenn Sie unsicher sind, fragen Sie Ihren Dozenten.

ein (neuer) Kapitän gewählt ist, soll diese Information dem Controller mitgeteilt werden.

Bitte beachten Sie, dass der Controller zwar eine Wahl anstoßen kann, an der Auswahl des Leaders aber nicht aktiv beteiligt ist!

Wie bei jeder anderen Aufgabe, definieren Sie hier neue Tests, die die Korrektheit der neuen MOM-Schnittstelle feststellen. Definieren Sie auch, wie die Performanz der Schnittstelle gemessen werden kann.

Wiederholen Sie die Performance-Tests aus Aufgabe 1 und 2 und vergleichen Sie die Testergebnisse, um festzustellen, wie die Performance aller Schnittstellen leidet, wenn gleichzeitig die MOM Schnittstelle unter Last steht. Dokumentieren Sie die dabei alle gewonnenen Erkenntnisse in einem neuen Messprotokoll und geben Sie es im Moodle ab.

4.5 Aufgabe 4 - Hochverfügbarkeit

Im letzten Schritt soll der Controller um Funktionalitäten zur Hochverfügbarkeit erweitert werden. Stellen Sie sicher, dass Ihr System den Ausfall eines Controllers verkraften kann. Hierbei sind Sie im Systemdesign und in der Auswahl der verwendeten Technologie frei.

Testen Sie Ihr System um sicherzustellen, dass Ausfälle verkraftet werden. Beschreiben Sie Ihr finales System sowie dessen Tests und deren Ergebnisse im Detail. Begründen Sie Ihre Technologieauswahl.

Wiederholen Sie die Performance-Tests aus Aufgabe 1, 2 und 3 und vergleichen Sie die Testergebnisse, um festzustellen, wie die Performance aller Schnittstellen variiert, wenn ein Controller ausfällt.

Dokumentieren Sie dabei alle gewonnene Erkenntnisse in Ihrem Messprotokoll und geben Sie es in Moodle ab.

5 Pi-Lab

In diesem Semester steht Ihnen das Pi-Lab (https://pi-lab.h-da.io/) zur Verfügung, um Ihre Anwendung in einem echten verteilten System zu deployen.

Das Pi-Lab besteht aus 8 Pi-Lab *Cubes*. Jeder Cube besteht wiederum aus 7 Raspberry Pi 4B *Knoten* mit jeweils 8 GB Arbeitsspeicher. Der Head-Knoten verfügt über ein Display. Die 6 Worker-Knoten sind Headless. Die Raspberry Pis werden mittels PXE gebootet und verfügen über keinen persistenten Speicher. Ein Neustart eines Knoten setzt den Pi somit wieder in seinen Ausgangszustand zurück. Alle Daten oder Änderungen gehen verloren.

Die Cubes wie auch einzelne Knoten lassen sich aus dem Netz der H-DA über eine Web-GUI starten und stoppen. Standardmäßig booten die Knoten eine modifizierte Version von Ubuntu 20.04 LTS Server, d.h. ohne GUI. Wenn ein Knoten

erfolgreich gestartet ist, können Sie sich über SSH mit Ihrem H-DA-Account auf dem Knoten einloggen.

Auf allen Knoten sind Git, Docker und Docker-Compose vorinstalliert. Sie können Ihre Anwendung somit relativ problemlos auf dem Pi-Lab deployen. Beachten Sie aber, dass es sich bei den Raspberry Pis um eine ARM-Architektur handelt. Eventuell müssen Sie Ihre Software (und die verwendeten Docker Images) an die Architektur anpassen bzw. auf dem Pi selbst kompilieren lassen.

Das Pi-Lab ist derzeit noch in der Beta-Phase. Wir freuen uns daher über eine rege Nutzung und konstruktives Feedback. Die effiziente Nutzung des Pi-Labs im Rahmen des Labors kann Sie für die Bonuspunkte qualifizieren. (siehe Abschnitt. 3). Sollten Sie darüber hinaus Interesse haben, z.B. im Rahmen einer Hiwi-Tätigkeit, einer Praxisphase oder einer Bachelorarbeit, an der Weiterentwicklung des Pi-Labs mitzuwirken, so sprechen Sie uns gerne direkt an.