Verteilte Systeme
Praktikum

Ronald Moore, Lars-Olof Burchard, Michael von Rüden, Stephan Gimbel

Hochschule Darmstadt
Fachbereich Informatik

Sommersemester 2023

1 Übersicht

Im Rahmen des Praktikums Verteilte Systeme soll eine Anwendung aus dem Bereich des *Distributed Computing / Financial Data Processing* entwickelt werden.

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.

Hintergrund: In 2023 hat eine *Bankenkrise* für Schlagzeilen gesorgt. Dabei wurden verschiedene Banken gerettet, unter anderem die Silicon Valley Bank und Credit Suisse. Besonders relevant für uns war die Rettung von der First Republic Bank in den USA, die von einer Gruppe anderer Banken gerettet wurde [1,3].

Als sie von dieser Ereignisse gehört haben, waren die Bewohner:innen der Insel-Staaten Lilliput und Blefuscu[1] gerade dabei, ein verteiltes System für deren noch nicht digitalisierten Banken im Auftrag zu geben. Nun ist es denen klar geworden, dass manchmal Banken einander retten sollen und, dass solche Rettungsaktionen von dem neuen System unterstützt werden müssen.

Also ist es Ihre Aufgabe, ein „Proof of Concept“ (eine Machbarkeitsstudie) für Lilliput und Blefuscu zu entwickeln. Genauer gesagt, sollen Se eine *verteilte Simulation* eines Systems entwickeln, das später für Lilliput und Blefuscu realisiert werden kann.

Dazu ist in mehreren Phasen jeweils ein Teil des in Abbildung 1 dargestelten Gesamtsystems zu erstellen.

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

[1] Die Inseln wurden zuerst in 1726 vom Jonathan Swift beschrieben [4]. Neuere Beschreibungen der Inseln finden Sie in [5] (auf Deutsch) oder in [8] bzw. [7] (auf Englisch).

Kleiner Hinweis: Die Inseln Lilliput und Blefuscu sind auch in der Rechnerarchitektur als die Geburtsorte der Begriffe „Little Endian“ und „Big Endian“ bekannt.

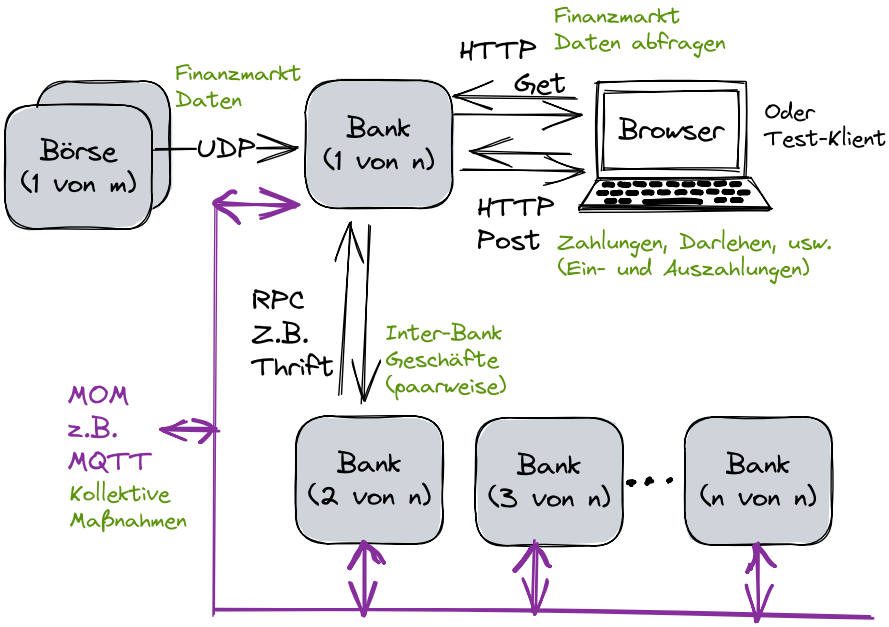

Abb. 1. Das Gesamtsystem.

(Deployen) der verteilten Anwendungen. Der Fokus liegt *nicht* auf einer fachlich korrekten Einführung in der Finanzmathematik oder der Finanzanalyse.

1.1 Grundregeln

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

- Das Bearbeiten der Praktikumsaufgaben findet so weit wie möglich in Zweier-Teams statt. Suchen Sie sich zu Beginn der Lehrveranstaltung einen verlässlichen Parter. Klären Sie im Vorfeld Ihre gegenseitigen Erwartungen.
- Die Praktikumsaufgaben müssen zu Hause vor- und nachbereitet werden, d.h. außerhalb der Praktikumstermine.[2]
- Die erste Aufgabe (UDP Sockets) muss spätestens beim 3. Gruppentermin korrekt und komplett vorgestellt (inkl. Testprotokoll) werden können. 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.

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

- Jedes Team 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 Master-Branch Ihres Repositorys zur Verfügung gestellt werden. Eigene Repositories sind nicht zulässig.
- Die Lösungen müssen containerisiert und mittels Docker und Docker-Compose zu testen sein. Nach Möglichkeit soll die Anwendung im Pi-Lab Cluster laufen; S. Absatz 3).
- 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.

Projektdokumentation Für jede Aufgabe, außer Aufgabe 0, muss ein Labor- bzw. Messprotokoll erstellt werden und ins Moodle hochgeladen werden. Jedes Team soll, für jede Aufgabe (1–4) selber entscheiden, wie am Besten die Leistung (Performance) des Systems zu messen ist. Die Messungen der vorherigen Aufgaben sollen immer wieder wiederholt werden, und auftretende Änderungen sollen beschrieben werden.

1.2 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 1-4) präsentieren, die deutlich über den Anforderungen und den Erwartungen liegt.

Manche Ideen und Vorschläge finden Sie unten, bei den einzelnen Aufgaben.

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 4.0 bestehen.

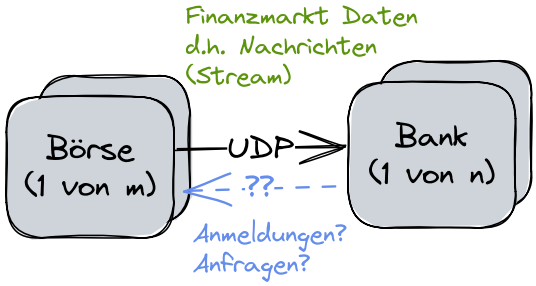

Abb. 2. Aufgabe 1 (UDP).

2 Aufgabenstellung

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.

Aufgabe 1 - UDP Sockets

Im ersten Schritt sollen zwei Komponenten realisiert werden: Börsen und Banken. Jede Bank hält eine simulierte Menge Wertpapiere, dessen Wert sich ständig ändert. Die Börse liefern einen Strom von Nachrichten, und zwar per UDP.[3]

Eine Nachricht besteht aus folgenden Komponenten: der Kürzel des Wertpapiers, das soeben gehandelt wurde, die Stückzahl und der Preis. Also könnte als Beispiel 50 Aktien vom Lillisoft (Kurzel: LSFT) für 280€ ver- bzw. gekauft sein.[4] Wenn die Erste Allgemeine Bank von Lilliput im Besitz von 1000 LSFT Aktien ist, sind sie nun 280.000€ wert. Wenn vorher der Preis von LSFT 300€ war, hat das Portfolio der Erste Allgemeine Bank einen Verlust von 20.000€.

[3] Die Verwendung von UDP ist nicht unbedingt sinnvoll oder vorbildhaft. Die ist aus pädagogischen Gründen motiviert. Wenn Sie einmal Erfahrung mit UDP-Sockets gemacht haben, können Sie besser beurteilen, ob das eine gute Entscheidung war.

[4] Jede Ähnlichkeit mit echten Wertpapieren wie z.B. Microsoft (MSFT) ist rein zufällig. Lilliput und Blefuscu haben zurzeit keine eigene Währung und verwenden Euro und Cent.

Der neue Gesamtwert des Portfolios soll nach Empfang jeden Nachrichten neu berechnet werden und im Text zum „Standard Out“ ausgegeben werden.

Hinweis: Sie sollen für Ihre Simulation eine Menge von Wertpapieren samt Kürzel und Anfangspreisen selbst definieren. Die Nachrichten sollen auch simulierte, d.h. frei erfundene Ereignisse beschreiben. Überlegen Sie, wie man das mit Zufallszahlen machen kann, sodass die Schwankungen der Kurse weder ganz wild und chaotisch sind, noch allzu langweilig bleiben. In der Finanzanalyse würde man sagen, Sie sollen für eine ausgemessene simulierte *Volatilität* sorgen.

Überlegen Sie, wie Sie Ihr System testen wollen. Sie sollen einerseits sicherstellen, dass das System *korrekt* ist. Darüber hinaus sollen Sie die *Leistung (Performanz)* Ihr System messen. Dafür müssen Sie entscheiden, was genau zu messen ist. Zum Beispiel kann *Round Trip Time* (RTT) nicht gemessen werden, solange Daten nur von den Börsen an den Banken übertragen werden. Also muss etwas anders gemessen werden — oder Daten auch in die andere Richtung fließen.

Werten Sie die Performanz Ihres Systems statistisch valide aus und erstellen Sie ein (sehr kurzes) Messprotokoll das den Test und die Ergebnisse darstellt.

Wenn Sie den Bonus (vgl. Absatz 1.2) erwerben wollen, überlegen Sie auch, ob die hier beschriebene Kommunikation wirklich ausreicht. Wie sollen die Börsen wissen, an wen Sie Nachrichten schicken sollen? Weiter, während manche Wertpapiere mehrmals per Sekunde gehandelt werden, werden andere ein oder zweimal *per Jahr* gehandelt. Wie soll eine Bank wissen, wie viel ein solches Wertpapier wert ist?

Aufgabe 2 - TCP Sockets und HTTP

Im nächsten Schritt sollen den Mitarbeiter:innen der Banken eine Web-Schnittstelle zur Verfügung gestellt werden. D.h. die Banken-Komponenten vom Aufgabe 1 sollen erweitert werden.- Die UDP-Schnittstelle bleibt erhalten und muss am Laufen bleiben.

Die neue Webschnittstelle hat zwei Hauptfunktionen:

1. Es soll möglich sein, der Gesamtwert der Bank zu erfahren. Das soll per HTTP GET erfolgen.
2. Weiter soll es möglich sein, neue Daten händisch einzugeben. Dass soll per HTTP POST erfolgen.

Diese Schnittstellen sind in Abbildung 3 dargestellt.

Die neuen Daten, die eingegeben werden können, sollen Sie selbst definieren. Gedacht sind hier Ein- und Auszahlungen, wenn Kunden Geld abholen oder anlegen, Kredite bekommen oder zurückzahlen, usw. usf. Die Mitarbeiter:innen dürfen auch Änderungen im Portfolio einer Bank machen, d.h. Wertpapier kaufen oder verkaufen.

Die Banken haben hiermit nun auch Geldreserven und ausstehende Krediten. Das geht auch in dem Gesamtwert der Banken ein. Das heißt, dass die Daten,

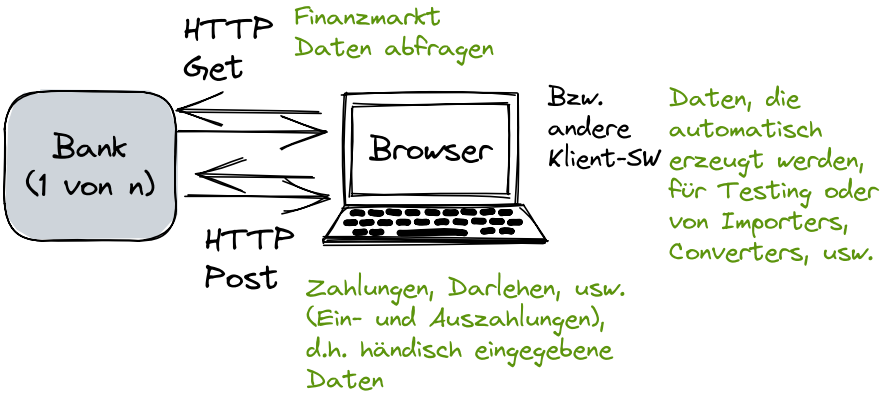

Abb. 3. Aufgabe 2 (TCP).

die per HTTP POST eingegeben werden, Einflüsse auf der Gesamtwert der Bank haben. Diese Ereignisse haben also Konsequenzen für die HTTP GET Schnittstelle und für die UDP Schnittstelle.

Die HTTP-Schnittstelle soll in einem normalen Browser, z.B. Firefox oder Google Chrome, vorgeführt werden.[5] Für die Performanz-Messungen können Sie auch einen eigenen Klient schreiben. Das wäre für die Bank sehr nützlich, falls sie später z.B. Daten automatisch importieren wollen. Es ist jedoch nicht streng notwendig, wenn Sie mit einem Browser die Performanz messen können.

Definieren Sie hier neue Tests, die die Korrektheit der neuen HTTP-Schnittstelle feststellen. Definieren Sie auch, wie die Performanz der Schnittstelle gemessen werden kann. Wiederholen Sie auch die Performance-Tests aus Aufgabe 1 und vergleichen Sie die Testergebnisse, um festzustellen, ob die Performance der UDP-Schnittstelle gelitten hat.

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

HTTP soll nur mit Hilfe von Sockets und ohne weitere Hilfsmittel (d.h. ohne externe HTTP-Bibliotheken) implementiert werden.[6] Es ist hierbei erforderlich,

[5] Sie müssen nur einen Browser unterstützen. D.h. entweder Firefox *oder* Google Chrome *oder* Internet Explorer, usw. usf. Sie müssen nicht unbedingt HTML verwenden, obwohl das für HTTP PUT zu empfehlen ist.

[6] 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, eine Kommunikationsprotokolle zu realisieren und, um Erfahrung mit TCP-Sockets zu bekommen.

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

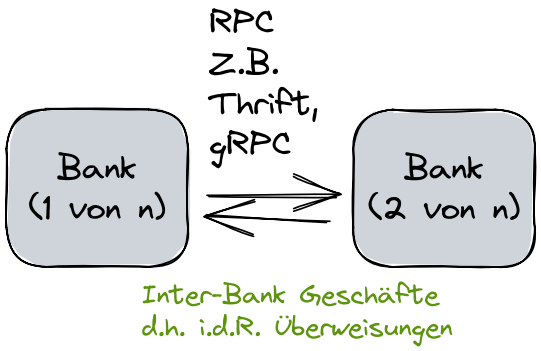

Abb. 4. Aufgabe 3 (Remote Procedure Calls (RPC)).

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.

Für den Bonus ist bunte HTML nicht unbedingt interessant; HTML und CSS werden in anderen Lehrveranstaltungen behandelt. Viel interessanter wäre eine ausgereifte Kommunikation, die die Arbeit der Mitarbeiter:innen erleichtert. Einer ausbaufähige eigene Importer- bzw. Converter-Klient wäre hier auch sinnvoll.

Aufgabe 3 - Remote Procedure Calls (RPC)

Bis jetzt wurden die Kommunikation zwischen Börsen und Banken in Aufgabe 1 simuliert, und in Aufgabe 2 die Kommunikation zwischen einer Bank und ihrer Mitarbeiter:innen simuliert. Nun sollen die Banken miteinander kommunizieren können, wie in Abbildung 4 zu sehen ist.

Die Banken sollen mittels Remote Procedure Calls (RCP) einander Dienste anbieten und diese Dienste in Anspruch nehmen. Dafür ist Apache Thrift zu empfehlen, aber auch Google's gRPC darf verwendet werden.[7]

Die Dienste sollen Sie definieren. Es soll möglich sein, dass eine Bank Geld von einer anderer Bank ausleiht. Darüber hinaus sollen Überweisungen unterstützt werden. Keine Bank muss jedoch eine Überweisung ausführen; alle Transaktionen können verweigert werden. Für Aufgabe 4 wird es wichtig sein, dass Überweisungen auch abgebrochen bzw. storniert werden können.

Ihre Simulation soll nun die Rettung einer Bank durch eine andere Bank simulieren können. D.h. einerseits, dass Sie in der Lage sein sollen, zu simulieren, was passiert, wenn eine Bank nicht mehr genug Geldreserve hat, im Fall eines

[7] Andere RPC-Frameworks sind (nur) nach Absprache mit dem Dozenten erlaubt.

Ansturms auf die Bank. Weiter, in so einem Fall will eine Bank Geld von einer anderen Bank leihen. Sie sollen so ein Szenario simulieren; sowohl wo die Bank gerettet wird, als auch wo die in Konkurs geht.

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.[8]

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 2 und vergleichen Sie die Testergebnisse, um festzustellen, ob die Performance der alten Schnittstellen gelitten hat.

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

Für den Bonus gibt es hier verschiedene Möglichkeiten. Zum Beispiel, eine *wirklich* gute, wohl überlegte RPC-Schnttstelle wäre geeignet. Auch gut wäre eine beeindruckende Simulation samt Aufzeichnungen, die eine Analyse ermöglichen. Besprechen Sie Ihre Pläne in jeden Fall mit Ihrem Dozent ab.

Aufgabe 4 - Message-oriented Middleware (MoM)

Endlich sind wir soweit, dass wir die Rettung einer gefährdeten Bank durch einer *Gruppe* von Banken simulieren können.

Hier sollen Sie Message-Oriented-Middleware (MOM) wie z.B. MQTT einsetzen.[9] Vgl. Abbildung 5.

Hier ist jede Bank sowohl ein *Publisher* als auch ein *Subscriber* von Nachrichten. Zuerst sollen die Banken regelmäßig Ihre Gesamtwerte veröffentlichen. Die andere Banken sollen diese Informationen verwenden, um Entscheidungen zu treffen. Besonders wichtig hier ist die Entscheidung, ob eine Bank eine andere Bank retten soll, wie in Aufgabe 3 schon simuliert wurde. Es soll also vorkommen, dass eine Bank nur durch *mehrere* andere Banken gerettet werden kann, weil keine andere Bank bereit ist, sie alleine zu retten.

Die Rettung erfolgt per angepasste Two Phase Commit (2PC). Das heißt, eine Menge Banken werden gefragt, ob sie bereit sind, eine Menge Geld an die

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

[9] Andere Middleware wie z.B. ZeroMQ oder AMQP sind (nur) auf eigenes Risiko und nach vorheriger Absprache mit Ihrem Dozenten erlaubt.

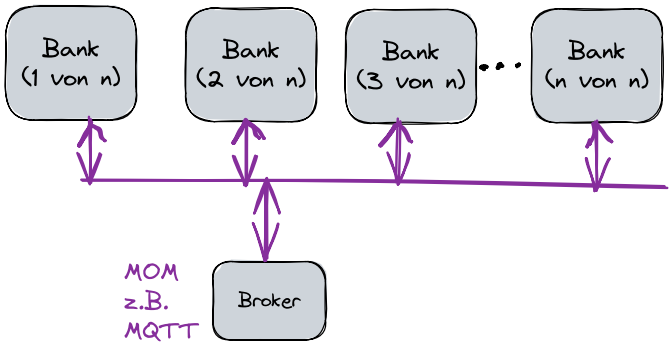

Abb. 5. Aufgabe 4: Message Oriented Middlware (MOM))z.B. MQTT.

gefährdeten Bank zu überweisen. Erst wenn ausreichend viele Banken zustimmen, werden die Überweisungen betätigt. Vgl. die Beschreibung von 2PC in der Vorlesung.

Hinweis: Der 2PC Algorithmus setzt voraus, dass eine Komponente den 2PC-Prozess (d.h. die Wahl) koordiniert. Sie sollen selbst definieren, wie entschieden wird, welche Bank der Koordinator ist. Für den Bonus könnte ein Konsens-Algorithmus hierfür verwendet werden, z.B. der Raft Algorithmus oder einer Paxos Algorithmus.[10]

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, 2 und 3 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 gewonnene Erkenntnisse in Ihrem Messprotokoll und geben Sie es im Moodle ab.

3 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

[10] Für Raft, s. [9] oder die Quellen dort. Für Paxos, s. [6] oder die Quellen dort. Nebenbei sei bemerkt, dass die Paxos Algorithmen nach Paxos, eine Insel in Griechenland, benannt wurden. Die Insel existiert also, auch wenn die dortige Verwendung der Algorithmen wohl fiktiv bleibt. Der Vergleich mit den Inseln Lilliput und Blefuscu können Sie an dieser Stelle selber ziehen. Für die Beziehung zwischen Rafts und Inseln bzw. eine andere lesbare Beschreibung des Raft Algorithmus, s. [2].

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 qualifiziert Sie für die Bonuspunkte (siehe Abschnitt. 1.2). 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.

Literatur

1. Bernau, P., Kremmer, D.: Wall Street banks will put $30 billion into beleaguered First Republic. Washington Post (2023), am 16. 03. 2023 erschienen; https://www.washingtonpost.com/us-policy/2023/03/16/yellen-bank-rescue-regional-banks/.
2. Mahn, J.: Wie verteilte Systeme dank Raft-Algorithmus zusammenarbeiten. heise online (2022), am 03. 10. 2022 erschienen; https://www.heise.de/hintergrund/Wie-verteilte-Systeme-dank-Raft-Algorithmus-zusammenarbeiten-7269322.html.
3. Stein, J.: Warum immer die Banken? Frankfurter Allgemeine Zeitung (2023), am 20. 03. 2023 aktualisiert; https://www.faz.net/aktuell/wirtschaft/svb-und-credit-suisse-warum-banken-uns-so-viele-sorgen-bereiten-18756740.html.
4. Swift, J.: Gulliver's Travels. Penguin Books (1726), reprinted in 2003. Also available at https://standardebooks.org/ebooks/jonathan-swift/gullivers-travels.
5. Wikipedia: Gullivers Reisen — Wikipedia, die freie Enzyklopädie. https://de.wikipedia.org/w/index.php?title=Gullivers_Reisen&oldid=231343776 (2023), [Online; Stand 2. April 2023]
6. Wikipedia: Paxos (Informatik) — Wikipedia, die freie Enzyklopädie. https://de.wikipedia.org/w/index.php?title=Paxos_(Informatik)&oldid=231565246 (2023), [Online; Stand 5. April 2023]
7. Wikipedia contributors: Gulliver's Travels — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Gulliver%27s_Travels&oldid=1145696051 (2023), [Online; Stand 2-April-2023]
8. Wikipedia contributors: Lilliput and blefuscu — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Lilliput_and_Blefuscu&oldid=1147243443 (2023), [Online; Stand 2-April-2023]

9. Wikipedia contributors: Raft (algorithm) — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Raft_(algorithm)&oldid=1146933760 (2023), [Online; Stand 5-April-2023]