Einsendeaufgabe 1

Architekturen und Prozesse

Wintersemester 2025/2026
31.10.2025

EINGEREICHT BEI
FernUniversität in Hagen
Fakultät für Mathematik und Informatik
Kooperative Systeme
Verteilte Systeme
Prof. Dr. Christian Icking

EINGEREICHT DURCH
Emily Lucia Antosch
emilyluciaantosch@web.de
Hamburg

Contents

1. Aufgabe 1.1

1.1. Aufgabe 1.1.1

Eine Ressource im Internet wird durch einen Universal Resource Locator (URL) identifiziert. Erklären Sie alle Informationen, die in

https://www.fernuni-hagen.de/mi/studium/ (1)

enthalten sind.

Lösung 1.1.1 Lösung 1

Die folgenden Infos lassen sich aus der

https://www.fernuni-hagen.de/mi/studium/ (2)

herauslesen:

- Das Protokoll ist **"https://"** (HyperText Transport Protocol Secure), welches für
  - Die Subdomain **"www"**, die eine Unterdomäne von "fernuni-hagen" ist.
  - Den Namen **"fernuni-hagen"** der Webseite, der mittels eines DNS dann in eine IP-Adresse umgewandelt wird
  - Die TLD(Top-Level-Domain) **".de"**, welche die Seite mit dem Land Deutschland verbindet/identifiziert.
  - Der Pfad **"/mi/studium/"**, welche den User auf eine Unterseite routet. Das trailing Slash deutet daraufhin, dass es sich um ein Verzeichnis handelt, was dann wahrscheinlich auf `index.html` weiterleitet.

1.2. Aufgabe 1.1.2

Inwiefern stellt die Verwendung von URLs eine Möglichkeit dar, Ortstransparenz zu erreichen? Warum bieten URLs mit Hilfe des DNS auch Replikationstransparenz?

Lösung 1.1.2 Lösung 2

Die Verwendung von URLs stellt eine Möglichkeit dar, Ortstransparenz zu erreichen, indem

1. die Ressourcen nicht nach geographischen Orten, sondern logisch geordnet werden,
2. die URL darüber hinaus Details des Netzwerks und dessen Topologie abstrahiert, muss sich der User nicht mit der IP-Adresse oder der Position des Servers beschäftigen und
3. die URL statisch und unveränderlich bleiben kann, auch wenn die Ressource selbst sich von einem Data Center zu einem anderen bewegt.

1.3. Aufgabe 1.1.3

Betrachten wir das E-Mail-System im Internet, wobei die Prozesse Mail User Agent und Message Transfer Agent (Mail-Server) daran beteiligt sind. Welche Prozesse gehören zur

Anwendungsschicht und welche zur Middleware-Schicht? Welche Prozesse sind zeitlich entkoppelt (temporally decoupled), aber referentiell gekoppelt (referentially coupled)?

Lösung 1.1.3 Lösung 3

- Der Prozess *Mail User Agent* ist Teil der Anwendungsschicht und ist referentiell gekoppelt, aber zeitlich entkoppelt. Der *Mail User Agent* ist direkter Teil der Anwendung, die dem User bereitgestellt wird. Um eine Nachricht zu senden, muss der *Mail User Agent* wissen, wohin die Nachricht gehen soll (E-Mail Adresse), da sie sonst nicht dahin kommt, wo sie hin soll. Ein *Mail User Agent*, sowohl beim Senden als auch beim Empfangen muss nicht gleichzeitig aktiv sein, da die Nachricht persistent in der Middleware gespeichert werden.
- Der Prozess *Mail Transfer Agent* ist Teil der Middleware-Schicht und ist referentiell gekoppelt, aber zeitlich entkoppelt. Der *Mail Transfer Agent* ist Teil der Middleware-Schicht, die sich darum kümmert, dass die Nachricht, die vom Sender kommt an den Empfänger gelangt. Um eine Nachricht weiterleiten zu können, muss der *Mail Transfer Agent* wissen, vom wem die Nachricht kommt und wohin die Nachricht gehen soll, weshalb der *Mail Transfer Agent* referentiell gekoppelt ist. Dabei muss der *Mail Transfer Agent*, solange eine andere Middleware die Nachricht zwischenspeichert, bis der *Mail Transfer Agent* bereit ist, nicht aktiv sein, wenn der Sender die Nachricht an den Empfänger schickt.

2. Aufgabe 1.2

Erklären Sie kurz, wie Publish-Subscribe-Systeme sich von (klassischen) eng gekoppelten Systemen unterscheiden.

Lösung 1.2 Lösung 4

Ein Publish-Subscriber Model definiert sich hauptsächlich darüber, dass der Publisher und der Subscriber stark entkoppelt sind. Der Publisher und der Subscriber kennen sich häufig nicht mal direkt. Der Publisher veröffentlich bei einem Event eine Nachricht und ein Subscriber abonniert bestimmte Nachrichten von Events. Die Kommunikation passiert über Middleware. Meistens müssen Subscriber und Publisher nicht gleichzeitig aktiv sein. In einem gekoppelten System kennen die Teilnehmer einander direkt. Kommunikation passiert über ein Protokoll direkt und beide Teilnehmer müssen zwangsläufig aktiv sein.

3. Aufgabe 1.3

3.1. Aufgabe 1.3.1

Was ist der Unterschied zwischen vertikaler Verteilung (engl. vertical distribution) und horizontaler Verteilung (engl. horizontal distribution)?

Lösung 1.3.1 Lösung 5

- Bei vertikaler Verteilung werden verschiedene Aufgaben eines System auf verschiedene Server verteilt. Dabei übernimmt dann eine Maschine eine bestimmte Aufgabe und eine andere Maschine eine völlig andere, wobei dann Informationen zwischen den Maschinen hin- und hergeschickt werden.
- Bei der horizontalen Verteilung werden die gleichen Aufgaben auf verschiedene Maschinen verteilt. So hat jede Maschine, die im verteilten System ist, die gleichen Aufgaben und der Load wird einfach über die Maschinen verteilt.

3.2. Aufgabe 1.3.2

In einem Peer-to-Peer-System werden Ressourcen von Hosts im Internet angeboten. Wozu benötigt man ein Overlay-Netzwerk für ein Peer-to-Peer-System?

Lösung 1.3.2 Lösung 6

In eine P2P-System sind beide Teilnehmer, also Peer 1 und Peer 2 gleichberechtigt und dienen sowohl als Client und Server. Häufig ist es in einem P2P-Netzwerk schwierig, andere Teilnehmer zu finden, eine Verbindung herzustellen oder eine Rechte-Struktur aufzubauen. Ein Overlay-Netwerk kann dann dabei helfen. Das Overlay-Netzwerk erlaubt effizientes Nachschlagen von möglichen Verbindungen und deterministisches Routing von Verbindungen.

3.3. Aufgabe 1.3.3

Was ist ein strukturiertes Overlay-Netzwerk?

Lösung 1.3.3 Lösung 7

Ein sturkturiertes Overlay-Netwerk ist ein Netzwerk, in dem eine deterministische Netzwerktopologie das Ansteuern von Nodes (Teilnehmer des P2P) unterstützt. Eine Netzwerktopologie kann zum Beispiel Ring oder Gitter sein. Dabei stellt das Netzwerk dann die Möglichkeit bereit, dass jeder Teilnehmer mittels einer Hash-Funktion angeprochen werden kann und über die logische Netzwerktopologie angeprochen werden kann.

3.4. Aufgabe 1.3.4

Beim Routing in einem strukturierten Overlay-Netzwerk werden die Nachrichten gemäß der logischen Verbindung von Peers gesendet. Ist der kürzeste Weg zwischen zwei Peers im Overlay-Netzwerk immer auch der kürzeste Weg zwischen ihnen im physischen Netzwerk? Begründen Sie Ihre Antwort.

Lösung 1.3.4 Lösung 8

Nein, es wird nicht zwangsläufig der kürzeste Weg zwischen zwei Peers im Overlay-Netwerk genutzt. Die physische Topologie ist zwar eine Variable, die die Topologie des Overlay-Netwerks bestimmen kann, aber es gibt auch Fälle, in dem die logische Verteilung von Maschinen im Overlay-Netzwerk dazu führt, das logisch weit entfernte Systeme, auch wenn sie direkt nebeneinander physisch liegen, logisch über mehrere Nodes geleitet werden.

3.5. Aufgabe 1.3.5

Wir betrachten ein System aus super peers und weak peers, das Anfragen zu Dateien beantworten kann.

- Welche Informationen soll der Index von super peers mindestens speichern?
- Welche Informationen muss ein weak peer beim Eintreten in das System seinem super peer mitteilen?
- Was muss ein super peer tun, wenn sich ein weak peer bei ihm abmeldet?

Lösung 1.3.5 Lösung 9

- Ein Index von Super Peers muss speichern, welche Daten es überhaupt gibt. Es muss aber auch speichern, wo diese Daten im eigenen Netzwerk von Weak Peers zu finden sind.
- Ein Weak Peer muss bei der Registrierung in einem System an den Super Peer melden, wie der Super Peer den Weak Peer erreichen können (Adresse). Und ein Weak Peer muss mitteilen, welche Daten über ihn erreichbar sind.
- Wenn ein Weak Peer das System verlässt, entweder mit Absicht oder durch einen Problem, muss der Super Peer den Weak Peer aus dem Index löschen, ihn aus der Liste der verbundenen Clients löschen und die Menge an verbundenen Clienten anpassen. Wenn der Weak Peer das System mit Absicht verlässt, muss er sein Verlassen beim Super Peer anmelden. Wenn er durch ein Problem offline geht, muss der Super Peer das feststellen und ihn auch aus dem System abmelden.

4. Aufgabe 1.4

Betrachten Sie das Python-Programm zur Client/Server-Kommunikation im Buch in Fig 2.3 (Seite 59, Distributed Systems, Third edition, Version 3.03 (2020)). Entwickeln Sie daraus ein Client/Server-Programm in Python über TCP, bei dem der Client durch “send time” nach dem Datum und der Uhrzeit fragt, der Server dies beantwortet, und der Client die Antwort zur Kontrolle ausgibt. Bringen Sie Ihr Programm auf Ihrem Computer zum Laufen! Hinweis: Server- und Client-Prozess können auf demselben Computer laufen, die für diesen Testzweck über die IP-Nummer 127.0.0.1 (localhost) kommunizieren dürfen. Wählen Sie selbst feste Portnummern. Python ist eine Programmiersprache, die immer mehr Bedeutung gewinnt und in der jeder Informatiker Erfahrungen gesammelt haben sollte.

Lösung 1.4 Lösung 10

Client Code 11

```python
from socket import socket, AF_INET, SOCK_STREAM

s = socket(AF_INET, SOCK_STREAM)

s.connect(("127.0.0.1", 8080))
msg = "send time"
s.send(msg.encode())
data = s.recv(1024)
print(data.decode())
s.close()
```

Server Code 12

```python
from socket import socket, AF_INET, SOCK_STREAM
import datetime

s = socket(AF_INET, SOCK_STREAM)
s.bind(("127.0.0.1", 8080))
s.listen(5)
(conn, addr) = s.accept()
while True:
    data = conn.recv(1024)
    if not data:
        break
    msg = data.decode()
    if msg == "send time":
        now = datetime.datetime.now().__str__()
        conn.send(now.encode())
conn.close()
```

5. Aufgabe 1.5

Wir betrachten einen Server mit einem Cache-Bereich in folgendem vereinfachten Modell: Der Server bekommt laufend Anfragen, die in 80 % der Fälle mit Hilfe des Cache in 5 ms beantwortet werden können. Im ungünstigeren Fall (20 %) ist ein langsamer Festplattenzugriff und deshalb zusätzlich 25 ms nötig, hier also 30 ms pro Anfrage.

- Wie viele Anfragen pro Sekunde kann der Server maximal beantworten, wenn er dafür einen einzigen Prozess ohne weitere Threads einsetzt?

Wir überlegen nun, ob es vorteilhaft ist, die Anfragen auf eine feste Zahl von mehreren parallelen Threads zu verteilen. Die Zeit zur Erzeugung, Umschaltung und beim Scheduling von Threads soll vernachlässigt werden.

- Wie viele Anfragen pro Sekunde kann der Server höchstens mit Hilfe von mehreren Threads beantworten?
- Wie viele Threads sollen sinnvollerweise hier vorgesehen werden?

Lösung 1.5 Lösung 13

Wenn der Server nur einen Thread benutzt, kann er maximal

$$1000 = x \cdot 0,8 \cdot 5 + x \cdot 0,2 \cdot 30 = x \cdot 4 + x \cdot 6 = x \cdot (4+6) = x \cdot 10 \rightarrow x = \mathbf{100} \quad (3)$$

Anfragen machen.

Mehrere Threads können durch die CPU-Zeit von 5 ms maximal

$$\frac{1000 \text{ ms}}{5 \text{ ms/req}} = \mathbf{200 \text{ req/s}} \quad (4)$$

machen.

Um diese Zeit auszureizen, müssen wir festlegen, dass

1. Alle CPU aktiv an etwas arbeiten (busy),
2. Wir genug Threads haben, um die maximale Anzahl an Requests zu erreichen,
3. Wir daher genug Threads für die I/O Wait Time haben.

Am Peak haben wir $200 \text{ req/s} \cdot 0,2\% = 40 \text{ req/s}$. Damit brauchen wir $40 \text{ req/s} \cdot 30 \text{ ms} = 1200 \text{ ms}$ für alle I/O-Operationen und $160 \text{ req/s} \cdot 5 \text{ ms} = 800 \text{ ms}$ für alle Cache-Operationen.

Da wir nur von Threads und nicht von CPU-Kernen sprechen, **müssen mindestens 3 Threads laufen**, um die maximale Anzahl von Requests zu machen. Zwei Threads schaffen es in einer Sekunde nicht, mit der Menge an Anfragen fertig zu werden. Bei 3 Threads kümmern sich zwei um I/O und einer um Cache. Vereinfacht schaffen zwei Threads nach 600 ms alle I/O-Threads zu beseitigen und der dritte Thread hat bereits

$$\frac{600 \text{ ms}}{30 \text{ ms/req}} \cdot 4 = 80 \text{ req} \quad (5)$$

erledigt. Die restlichen 400 ms reichen dann für die restlichen $80 \text{ req} = \frac{400 \text{ ms}}{5 \text{ ms/req}}$ aus.