- **Verteilte Systeme (VS)**
- **Zeitpunkt:** September 2013
- **Dauer:** 20 Minuten (5 ECTS)
- **Prüfer:** Jürgen Kleinöder
- **Beisitzer:** Tobias Distler
- **Ergebnis:** sehr gut, faire Benotung

Vorbemerkung
Entspannte Atmosphäre. Eher ein Prüfungsgespräch, indem sich der Student selbst von Thema zu Thema hangelt, als ein Frage-Antwort-Spiel. Gesprächsrichtung kann unter Umständen beeinflusst werden!
Zügiges Vorankommen durch den „Lernstoff" ist empfehlenswert. Dadurch bleibt dem Prüfer Zeit bestimmte Aspekte zu vertiefen. Sollte man da mal etwas aus dem Tritt kommen, bloß nicht verunsichern lassen! Vonseiten des Prüfers wird da auch gerne helfend eingegriffen.

Was versteht man unter einem Verteilten System?
Mehrere unabhängige Rechner, verbunden durch ein Netzwerk, die dem Anwender als Einzelrechner erscheinen (Transparenzen) und kooperativ eine Aufgabe lösen. Kommunikation erfolgt durch Nachrichtenaustausch, kein gemeinsamer Speicher! Transparenzen verbergen Verteiltes System weitestgehend vor dem Anwender, auch bei Fehlern.

Wie könnte denn Fehlerbehandlung in Verteilten Systemen aussehen?
Redundanz, also z.B. redundante Leitungen oder Replikate.
Einsatz zuverlässiger Transportprotokolle: TCP statt UDP.
Fehlererkennung durch Prüfsummen. Sendewiederholungen von Nachrichten, hierbei Einsatz von speziellen Semantiken.

Welche Semantiken kennen Sie denn und Semantiken wofür?
RPC-Semantiken, also Semantiken für Prozedurfernaufrufe. Sprich ein Prozess will eine Aktivität auf einem entfernten Prozess ausführen.

- × Maybe
- × At-Least-Once
- × At-Most-Once
- × Last-Of-Many
- × Exactly-Once

Mit Maybe besitzt der Sender im Fehlerfall keine Garantie der Ausführung auf Empfängerseite.
Mit At-Least-Once werden unter Umständen Sendewiederholungen, nach Ablauf eines Timeouts, auf Empfängerseite mehrfach ausgeführt. Empfänger muss daher idempotente Operationen bereitstellen. Idempotente Operationen bedeuten: Gleiches Ergebnis bei exakt gleichem Aufruf (gleicher Aufrufer, gleicher Aufgerufener, gleiche Aufrufparameter) **UND** gleicher Ausgangszustand auf Empfängerseite nach Ausführung.

Gibt es auch idempotente Schreiboperationen?
Lesen funktioniert immer. Schreiben nur wenn Empfängerzustand unverändert bleibt, z.B. bei „Setze"-Operationen. Inkrementelle Schreiboperationen sind ein Problem (Liste erweitern / Zähler erhöhen).

Wie könnte man unser Bankbeispiel mit idempotenten Schreiboperationen lösen?
Erster Gedanke: transaktionsähnlicher Ablauf. Klient liest eigenen Kontostand und verändert ihn lokal. Danach überprüft er aktuellen Kontostand auf Empfängerseite und schreibt ihn nur zurück, wenn sich dieser seither nicht verändert hat. Allerdings ist Überprüfen und Zurückschreiben nicht atomar, geht also nicht.
Zweiter Gedanke: kritischer Abschnitt. Klient fordert Lock an und kann somit unbeeinflusst Kontostand verändern.

Wie ist das dann mit Locks und idempotenten Operationen?
Problemsituation: Klient fordert Lock an (idempotente Operation). Antwortnachricht geht allerdings verloren. Klient wiederholt Lockanforderung. Server muss somit erkennen können, dass der entsprechende Klient das Lock eigentlich schon besitzt!

Welche Semantiken gibt es noch?
At-Most-Once-Semantik. Sender verfügt über Garantie, dass Ausführung höchstens 1x stattfindet. Empfänger muss daher Sendewiederholungen eindeutig identifizieren können. Empfänger speichert sich nach erstmaliger Ausführung Ergebnis und schickt bei Sendewiederholungen entsprechendes Ergebnis zurück. Sender wiederholt Anfragen z.B. bis zu einem definierten Maximalwert.

Wann könnte man denn diese gespeicherten Ergebnisse wieder freigeben?

- implizit: sobald neue Nachricht vom Sender empfangen wird
- explizit: durch einen speziellen Aufruf vonseiten des Senders
- Timeout: immer als Rückfallebene möglich

OK. Bleiben wir mal beim Timeout. Angenommen ein Sender führt alle Sendewiederholungen durch, der Timeout auf Empfängerseite ist abgelaufen und es kommen noch Wiederholungen an. Kann das überhaupt passieren, wäre das überhaupt ein Problem?
Ja. Nachrichten können verzögert werden. Das ist ein Problem. Beim Eintreffen verzögerter Nachrichten können die dazugehörigen Ergebnisse aufgrund des Timeouts bereits gelöscht worden sein. Somit wird unter Umständen die At-Most-Once-Semantik verletzt!

Ich behaupte, ein Empfänger kann unter anderem mit Hilfe eines Timeouts feststellen, wann auf jeden Fall zwischengespeicherte Ergebnisse löschen kann.
So 100% krieg ichs leider nicht mehr hin. Der Ansatz war mit physikalischen Zeitstempeln in den Sendenachrichten zu arbeiten unter der Annahme synchronisierter Uhren (z.B. mit NTP). Man könnte bspw. dem Empfänger die maximal Anzahl an Sendewiederholungen sowie den Wiederholungstimeout der Senderseite mitteilen. Daraus lässt sich auf Empfängerseite das Zeitintervall ermitteln, für das Nachrichten akzeptiert werden. Erreicht den Empfänger bspw. dann eine verzögerte Nachricht aus diesem Intervall und wurde entsprechende Anfrage bereits 1x ausgeführt die Ergebnisse allerdings aufgrund des Timeouts bereits gelöscht, so könnte er anhand des Zeitintervalls feststellen, dass entsprechende Anfrage wahrscheinlich schon mal ausgeführt wurde. So oder so ähnlich ☺.