

Verteilte Systeme 2

Prinzipien und Paradigmen

Hochschule Karlsruhe (HsKA)
Fakultät für Informatik und Wirtschaftsinformatik (IWI)
christian.zirpins@hs-karlsruhe.de

Kapitel 01: Einführung

Version: 9. Oktober 2018



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Inhalt

01: Einführung
02: Architekturen
03: Prozesse
04: Kommunikation
05: Benennung
06: Koordination
07: Konsistenz & Replikation
08: Fehlertoleranz
09: Sicherheit

Einordnung in der Fakultät IWI (Informatik)

Dozent: Prof. Dr. rer. nat. Christian Zirpins

Seit 2015 Professor für verteilte Systeme an der HsKA

Bereich Verteilte Systeme (VSYS)

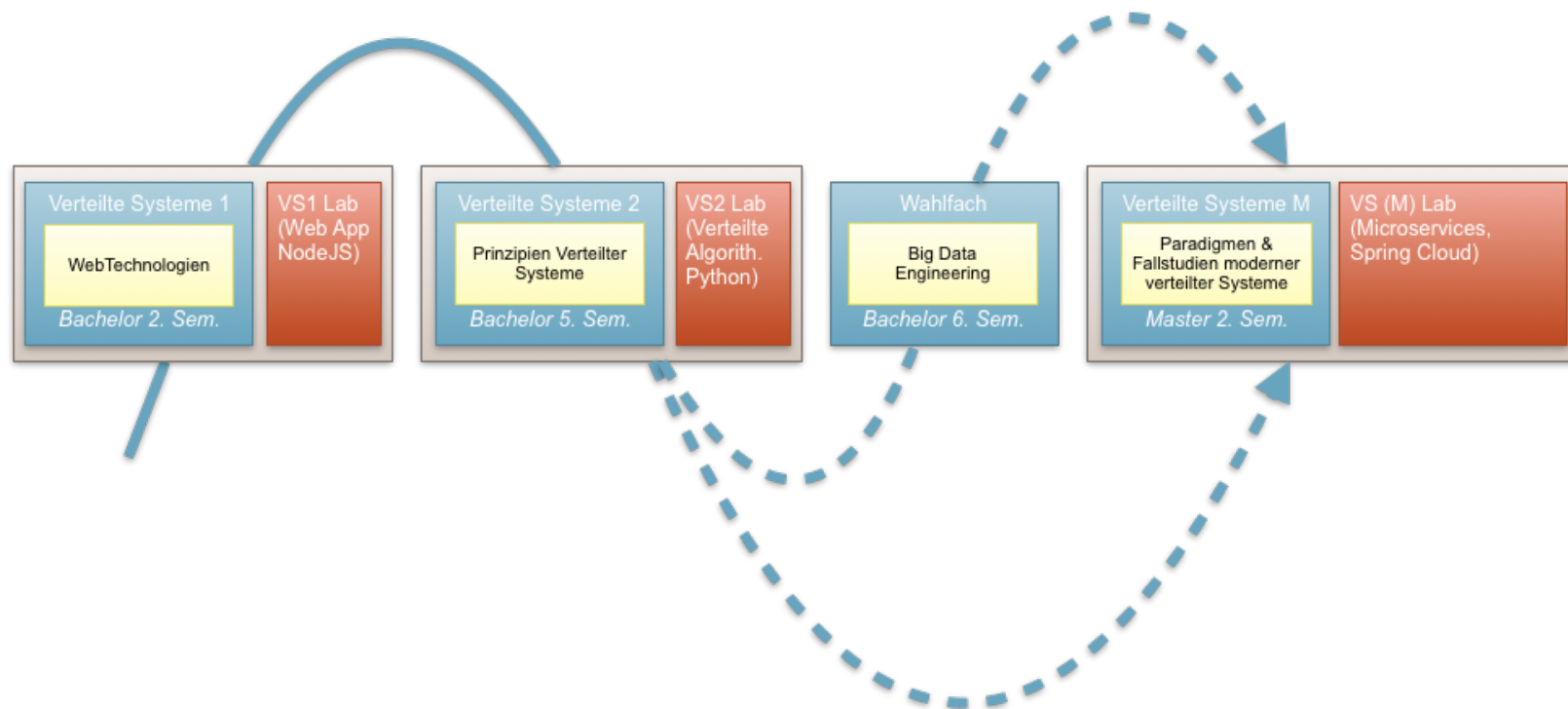
Schwerpunkte in Forschung und Lehre:

- Web Engineering
- Datenintensive Systeme (“Big Data”)
- Service Computing (“Microservices”)

Mit Bezügen zu...

Cloud Computing, Internet of Things, Smart Systems u.a.

Lehrpfade “Verteilte Systeme” an der HsKA



VS2 Termine im Wintersemester 2018

KW	Datum	Vorlesung (jeweils Mi. 08:00 Uhr in E 311)
40	10.10.	Einführung
41	17.10.	Architekturen
42	24.10.	Architekturen / Prozesse
43	31.10.	Prozesse / Kommunikation
44	07.11.	Kommunikation
45	14.11.	Benennung
46	21.11.	Koordination
47	28.11.	Koordination
48	05.12.	Replikation und Konsistenz
49	12.12.	Replikation und Konsistenz
50	19.12.	Fehlertoleranz
51	26.12.	Entfällt (Weihnachtsferien)
52	02.01.	Entfällt (Weihnachtsferien)
01	09.01.	Fehlertoleranz
02	16.01.	Sicherheit
03	23.01.	Sicherheit
26	30.01.	Outro / Q&A

VS2 Labor im Wintersemester 2018

Verteilte Systeme 2 Labor

- **Vor** SPO7: freiwillige Übung mit Klausurbonus
- **Ab** SPO7: Pflichtveranstaltung im Modul mit Bonusaufgabe
- Erster Termin nächste Woche

KW	Datum	Präsenzteil (jeweils Do. 14:00 Uhr in Li 137)
41	18.10.	Mehrschicht-Architekturen mit Sockets
44	08.11.	Kommunikation per Remote Procedure Call (RPC)
46	22.11.	AKommunikation über Nachrichten mit ZeroMQ
48	06.12.	Namensauflösung im Chord P2P-System
50	20.12.	Mutex-Koordination mit logischen Lamport Uhren
02	17.01.	Fehlertoleranz mit 2-Phasen Commit Protokoll
04	31.01.	Letzte Abgabe

Ressourcen und Support

Foliensätze / Begleitmaterial etc. auf ILIAS

- Anmeldung mit Passwort (pwd 'ic4ip')
- `http://bit.ly/2d018JS`

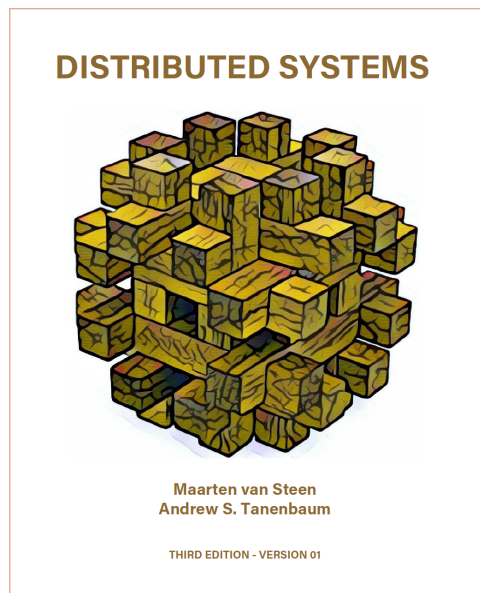
Email Kontakt

- Individuelle Fragen werden gerne per Email beantwortet.
- `christian.zirpins@hs-karlsruhe.de`

Literatur

Begleitbuch / Skript

Marten van Steen, Andrew S. Tanenbaum, “Distributed Systems”, 3rd edition, CreateSpace Independent Publishing Platform, 2017, ISBN 1543057381¹



Foliensatz

Basierend auf der Vorlesung
“Distributed Systems” von Maarten
van Steen an der VU Amsterdam.

¹Freie digitale Version: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017/>

Verteiltes System

Definition

Ein Verteiltes System ist eine Sammlung **autonomer Rechenelemente**, die ihren Nutzern wie ein **einzelnes kohärentes System** erscheint.

Charakteristische Eigenschaften

- Autonome Rechenelemente, auch **Knoten** genannt, ganz gleich ob Hardwaregeräte oder Softwareprozesse
- Einzelnes kohärentes System: Nutzern oder Anwendungen erscheint es als ein System \Rightarrow Knoten müssen **zusammenarbeiten**.

Sammlung autonomer Knoten

Unabhängiges Verhalten

Jeder Knoten ist autonom mit **eigenem Zeitbegriff**: es gibt keine **globale Uhr** – führt zu fundamentalen Synchronisations- und Koordinationsproblemen.

Sammlung von Knoten

- Wie verwaltet man **Gruppenmitgliedschaft**?
- Wie kann man wissen, ob man mit einem **autorisierten (Nicht-)Mitglied** kommuniziert?

Kohärentes System

Im Kern

Die Sammlung von Knoten als Ganzes arbeitet gleich, egal wo, wann und wie die Interaktion von Nutzer und System erfolgt.

Beispiele

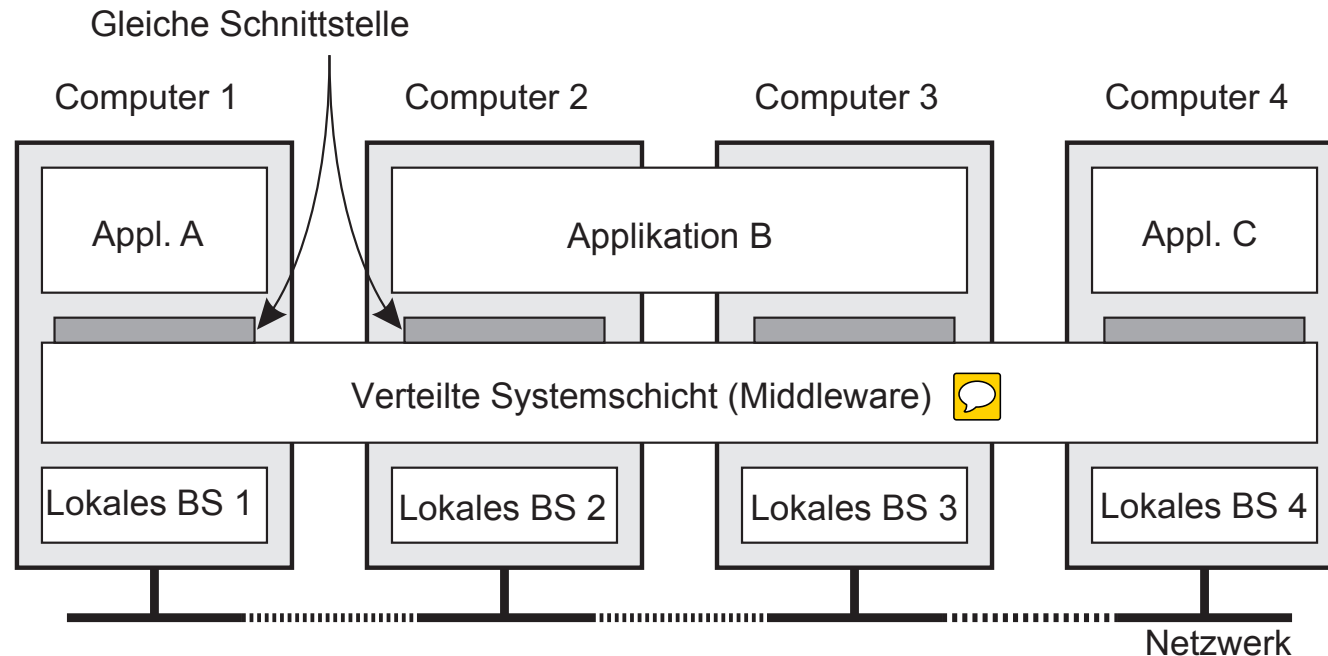
- Anwender können nicht sagen, wo Berechnungen stattfinden
- Speicherort von Daten sollte für Anwendungen irrelevant sein
- Ob Daten repliziert wurden oder nicht ist komplett versteckt

Der Schlüssel ist **Verteilungstransparenz**

Problematisch: partielle Ausfälle

Unvermeidbar, dass manchmal **ein Teil** des Systems ausfällt. Partielle Ausfälle zu verstecken und zu beheben ist oft sehr schwer und z.T. unmöglich.

Middleware: das Betriebssystem verteilter Systeme



Was gehört dazu?

Häufig genutzte Komponenten und Funktionen, die nicht von Anwendungen separat implementiert werden brauchen.


Was wollen wir erreichen?

- Teilen von Ressourcen
- Verteilungstransparenz
- Offenheit
- Skalierbarkeit

Ressourcen teilen

Typische Beispiele



- Geteilter Speicher und Dateien in der Cloud
- Peer-to-Peer-gestütztes Multimedia Streaming 
- Geteilte Email Services (z.B. Outsourcing des Email Systems)
- Geteiltes Web Hosting (z.B. Content Distribution Networks)



Beobachtung

“The network is the computer”

(Zitat John Gage, damals bei Sun Microsystems)

Verteilungstransparenz

Typen

Transparenz	Beschreibung
Zugriff	Verberge Unterschiede in Datenrepräsentation und Aufrufmechanismen
Ort	Verberge, wo ein Objekt sich befindet
Relokation	Verberge, dass ein Objekt während seiner Nutzung an einen anderen Ort bewegt wird 
Migration	Verberge, dass ein Objekt seinen Ort wechselt
Replikation	Verberge, dass ein Objekt repliziert ist
Nebenläufigkeit	Verberge, dass ein Objekt von Nutzern geteilt wird 
Ausfall	Verberge Ausfall und Wiederherstellung eines Objekts



Offenheit verteilter Systeme

Was ist damit gemeint?

Mit Diensten anderer offener Systeme interagieren können, unabhängig von der zugrundeliegenden Umgebungen:


- Systeme haben wohldefinierte **Schnittstellen**
- Systeme sollten einfach **interagieren**
- Systeme sollten **Portabilität** von Anwendungen erlauben
- Systeme sollten einfach **erweiterbar** sein

Skalierung im verteilten Systemen

Beobachtung

Entwickler moderner verteilter Systeme sprechen z.T. von “skalierbar”, ohne klarzustellen, **wie** ihr System eigentlich skaliert.

Mindestens drei Dimensionen

- Anzahl der Nutzer und/oder Prozesse (größenmäßig skalierbar)
- Maximale Distanz zwischen Knoten (geographisch skalierbar)
- Anzahl administrativer Domänen (administrativ skalierbar) 

Beobachtung

Viele Systeme berücksichtigen nur größenmäßige Skalierbarkeit mittels mehrerer starker Server, die unabhängig parallel arbeiten. Krux liegt in geographischer und administrativer Skalierbarkeit.

Größenmäßige Skalierbarkeit

Ursachen von Skalierungsproblemen zentralisierter Lösungen

- Rechenkapazität, limitiert durch CPUs
- Speicherkapazität, inkl. Transferrate CPUs zu Platten
- Netzwerk zwischen Nutzer und zentralisiertem Dienst

Probleme mit geografischer Skalierbarkeit

- Von LAN auf WAN wechseln ist schwierig: viele verteilte Systeme verlassen sich auf **synchrone Client-Server Interaktionen**: Clients senden Requests und warten auf Antwort. **Latenz** im WAN macht dies schwer.
- WAN Verbindungen sind oft **unzuverlässig**: einen Videostream einfach vom LAN auf das WAN zu verschieben ist problematisch.
- **Multipoint-Kommunikation fehlt**, so dass Suche per Broadcast ausscheidet. Stattdessen sind spezielle **Namensdienste** nötig.

Probleme mit administrativer Skalierbarkeit

Im Kern

Konflikte von Richtlinien für Nutzung (Zahlung), Management und Sicherheit

Beispiele

- **Hochleistungsrechner/Grids**: teile teure Ressourcen zwischen verschiedenen Domänen.
- **Geteiltes Equipment**: wie soll ein geteiltes Radioteleskop, das als großes Sensornetzwerk konstruiert ist, gesteuert, verwaltet und genutzt werden?


Ausnahme: einige Peer-to-Peer Netze

- File-Sharing Systeme (basierend, z.B., auf BitTorrent)
- Peer-to-Peer Telefonie (Skype)
- Peer-Assisted Audio Streaming (Spotify)

Merke: **Endanwender** kollaborieren und nicht **administrative Einheiten**.


Techniken der Skalierung

Verberge Kommunikationslatenz

- Nutze **asynchrone Kommunikation** 
- Separate “Handler” für eingehende Antworten
- **Problem:** nicht für alle Anwendungen passend

Techniken der Skalierung

Partitioniere Daten und Berechnungen über mehrere Maschinen

- Verschiebe Berechnungen zu Clients (Java Applets) 
- Dezentralisierte Namensdienste (DNS)
- Dezentralisierte Informationssysteme (WWW)

Techniken der Skalierung

Replikation/Caching: Mache Kopien von Daten auf verschiedenen Maschinen verfügbar


- Replizierte Dateiserver und Datenbanken
- Gespiegelte Web Sites
- Web Caches (in Browsern und Proxies)
- Datei Caching (bei Server und Clients)

Skalierung: Das Problem der Replikation

Anwendung von Replikation ist einfach bis auf Eines

- Mehrere Kopien (gecached oder repliziert) führen zu **Inkonsistenz**: Modifikation einer Kopie macht diese unterschiedlich zum Rest.
- Um Kopien auf generelle Weise immer konsistent zu halten, wird **globale Synchronisation** bei jeder Modifikation nötig.
- Globale Synchronisation schließt umfangreiche Lösungen aus.

Beobachtung

Können wir Inkonsistenz tolerieren, kann globale Synchronisation reduziert werden, aber **Toleranz von Inkonsistenzen ist anwendungsabhängig**. 

Übung 1: Skalierbarkeit

Use Case Analyse

Auf den folgenden Folien werden drei Fälle beschrieben, bei denen sich die verschiedenen Dimensionen der Skalierung zeigen und verschiedene Skalierungstechniken zur Anwendung kommen.

Geben Sie für jeden Fall an ...

- ... um welche Dimension der Skalierung es sich handelt und
- ... welche Skalierungstechnik zum Einsatz kommt.

Drei Beispiele für Skalierung (Fall 1)

Dimensionen: *größenmäßig, geografisch, administrativ*

Techniken: *verbergen, partitionieren, replizieren*

Kreditkartentransaktionen

Ein Finanzunternehmen bietet die Abwicklung von Kreditkartentransaktionen als Online-Service für Web-Shops an.

Da bei großer Nachfrage und entsprechender Last **eine Wartezeit** auftreten kann, erfolgt die Bestätigung einer Zahlung **nicht unmittelbar** auf der Benutzeroberfläche sondern **später per Email**.

Drei Beispiele für Skalierung (Fall 2)

Dimensionen: *größenmäßig, geografisch, administrativ*

Techniken: *verbergen, partitionieren, replizieren*

Software-as-a-Service

Ein Software-as-a-Service Anbieter möchte sein Angebot von Deutschland aus auf Nordamerika ausweiten.

Um die Latenz für US-Nutzer zu verbessern, mietet er virtuelle Maschinen eines Cloud Providers in dessen US-Zone an.

Auf den virtuellen Maschinen erstellt er exakte Kopien des bisherigen Anwendungsservers und der Datenbank.

Drei Beispiele für Skalierung (Fall 3)

Dimensionen: *größenmäßig, geografisch, administrativ*


Techniken: *verbergen, partitionieren, replizieren*

Web-Marktplatz

Ein Web-Marktplatz soll offen sein, die Angebote neuer Handelsunternehmen zu integrieren.

Um die individuellen Sicherheitsregeln der beteiligten Unternehmen zu unterstützen, bekommt jeder Händler einen exklusiven Server mit eigener Produktdatenbank, die vom zentralen Marktplatz aus angesprochen wird.

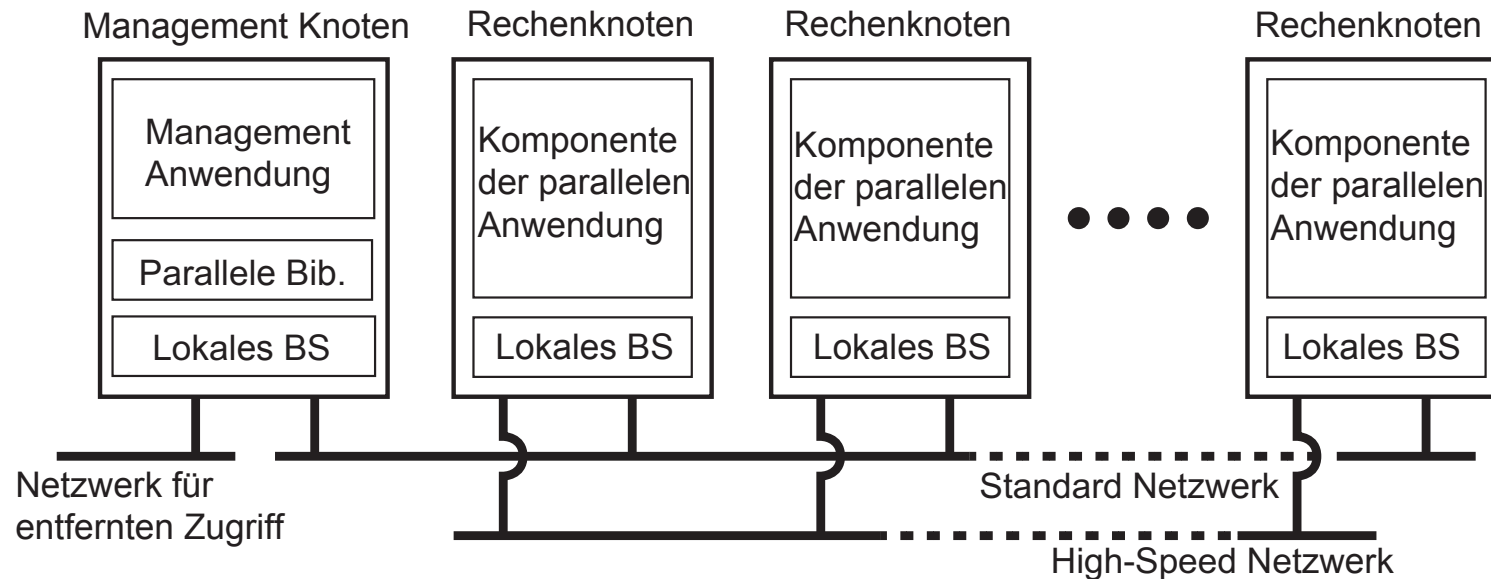
Typen verteilter Systeme

- Verteilte Hochleistungsrechnensysteme
- Verteilte (betriebliche) Informationssysteme
- Verteilter Systeme für *Pervasive Computing* 

Cluster Computing

Gruppe von High-End Systemen verbunden per LAN

- Homogen: gleiches BS, nahezu identische Hardware
- Einzelner Managementknoten



Grid Computing

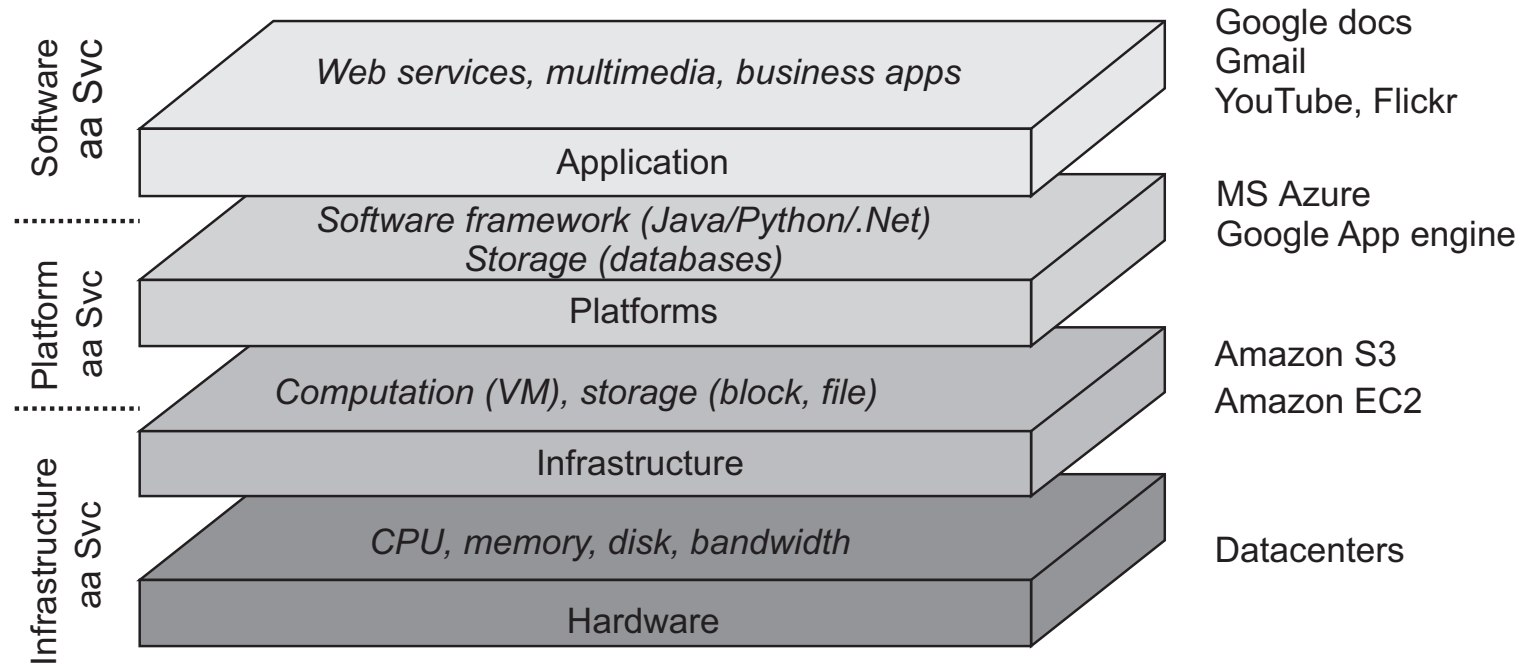
Der nächste Schritt: viele Knoten von überall 

- Heterogen
- Verstreut über vielfältige Organisationen
- Können leicht ein Wide-Area Network bilden

Anmerkung

Um Kollaborationen zu erlauben, nutzen Grids **virtuelle Organisationen**. Im wesentlichen ist das eine Gruppe von Nutzern (bzw. ihren IDs), die die Autorisierung von Ressourcenallokation ermöglichen.

Cloud Computing



Cloud Computing

Unterscheidet mindestens vier Schichten:

- **Hardware:** Prozessoren, Router, Strom- und Kühlsysteme. Kunden sehen diese normalerweise nie.
- **Infrastruktur:** Nutzt Techniken der Virtualisierung. Basiert auf Zuordnung und Management virtueller Speichergeräte und virtueller Server.
- **Plattform:** Erbringt höhere Abstraktionen für Speicher usw. Beispiel: Amazon S3 (Simple Storage Service) bietet eine API für (lokal erzeugte) Dateien, die in **Buckets** organisiert und gespeichert werden.
- **Anwendung:** Programme und GUIs für Endanwender, z.B. Textverarbeitung, Tabellenkalkulation etc.

Enterprise Application Integration (EAI)

Situation

Unternehmen betreiben meist viele **vernetzte Anwendungen**, aber Interoperabilität zu erreichen ist nicht leicht.

Grundlegender Ansatz

Eine vernetzte Anwendungen läuft auf einem **Server** und bietet ihre Dienste entfernten **Clients** an. Einfache Integration: Clients kombinieren Anfragen an (verschiedene) Anwendungen; senden sie ab; sammeln die Antworten und präsentieren dem Nutzer ein kohärentes Ergebnis.

Nächster Schritt

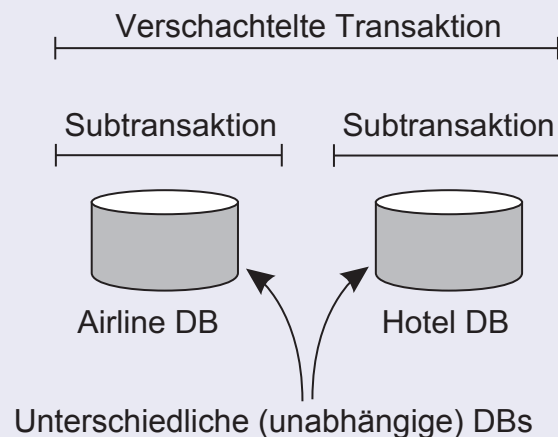
Direkte Kommunikation von Anwendung zu Anwendung ermöglichen, was zu **Enterprise Application Integration** (EAI) führt.

Beispiel EAI: (verschachtelte) Transaktionen

Transaktion

Primitive	Beschreibung
<i>BEGIN_TRANSACTION</i>	Starte Transaktion
<i>END_TRANSACTION</i>	Beende Transaktion, versuche Abschluss
<i>ABORT_TRANSACTION</i>	Breche Transaktion ab, setze Werte zurück
<i>READ</i>	Lese Daten aus Datei, Tabelle etc.
<i>WRITE</i>	Schreibe Daten in Datei, Tabelle etc.

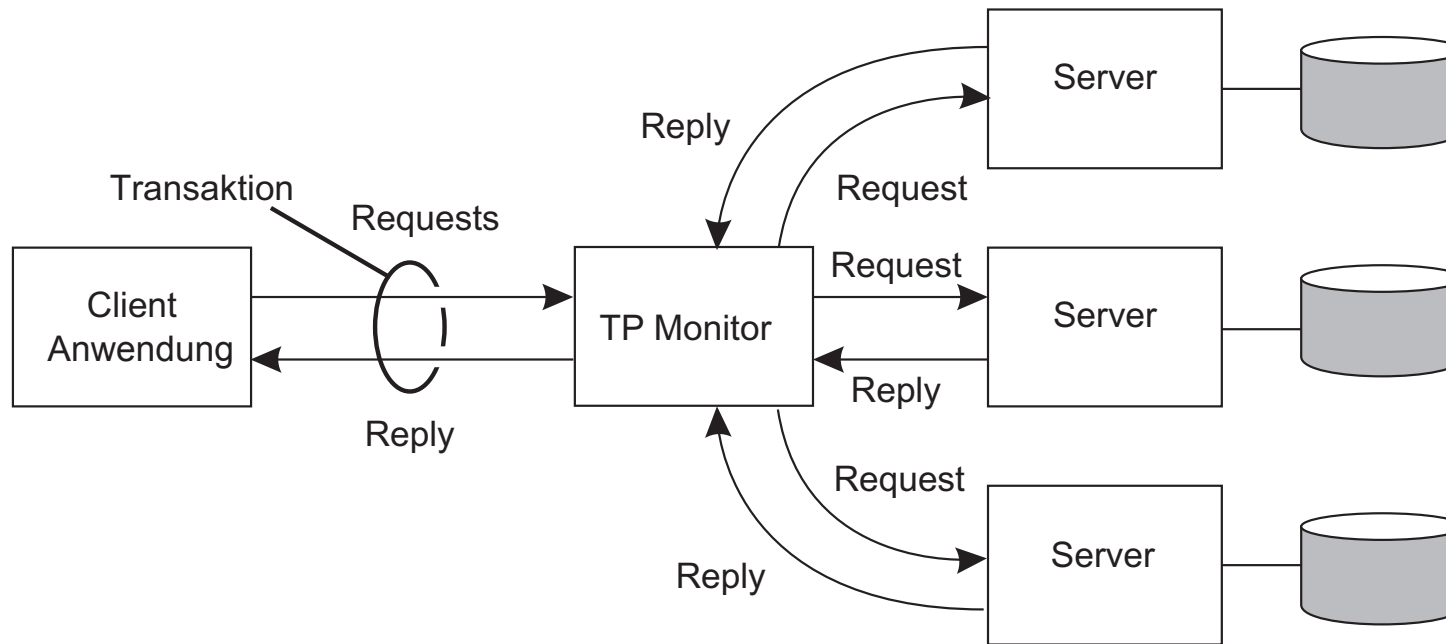
Wesentlicher Punkt: alles oder nichts



ACID Eigenschaften

- **Atomic:** passiert untrennbar (scheinbar)
- **Consistent:** respektiert Invarianten
- **Isolated:** keine wechselseitige Störung
- **Durable:** Abschluss/Commit ist dauerhaft

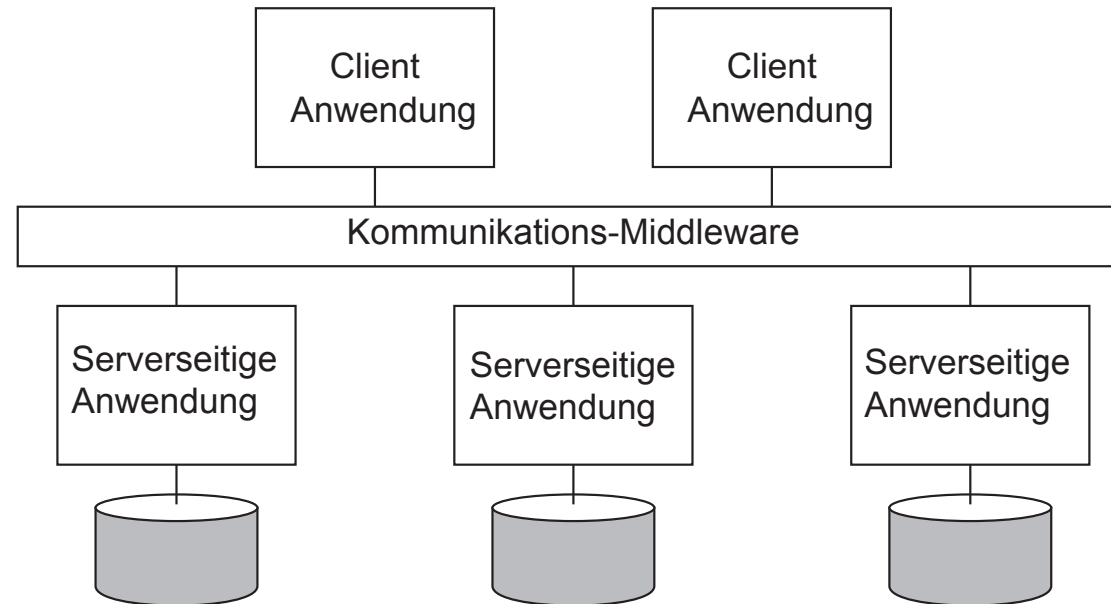
TPM: Transaction Processing Monitor



Beobachtung


In vielen Fällen sind die Daten einer Transaktion über mehrere Server verteilt. Ein **TP Monitor** koordiniert die Ausführung einer Transaktion.

Middleware und EAI



Middleware bietet Kommunikationsmittel zur Integration

Remote Procedure Call (RPC): Sendet Anfragen durch lokale Prozeduraufrufe, gibt Antworten als Rückgabe der Aufrufe zurück.



Message Oriented Middleware (MOM): Sendet (**publiziert**) Nachrichten an logischen Kontaktpunkt, Weiterleitung an **abonnierte** Anwendungen. 

Verteilte pervasive Systeme

Beobachtung


Nächste Generation verteilter Systeme, bei denen Knoten klein, mobil und oft in größere Systeme eingebettet sind. Charakterisiert dadurch, dass die Systeme sich **natürlich in die Umgebung des Nutzers einfügen**.

Drei (überlappende) Subtypen

- **Ubiquitäre Systeme**: allgegenwärtig und **ständig präsent**, d.h., es gibt einen ständigen Austausch zwischen System und Nutzer. 
- **Mobile Systeme**: allgegenwärtig aber Betonung auf **inhärenter Mobilität** der Geräte. 
- **Sensor (und Aktor) Netze**: allgegenwärtig, mit Betonung auf (kollaborativem) **Messen** und **Regeln** der Umwelt.


Ubiquitäre Systeme

Kernelemente

- 1 (**Verteilung**) Geräte sind vernetzt, verteilt und transparent zugreifbar
- 2 (**Interaktion**) unaufdringliche Interaktion von Nutzer und Gerät 
- 3 (**Kontextsensitivität**) Gerät erkennt Nutzerkontext und optimiert Interaktion
- 4 (**Autonomie**) Geräte laufen autonom ohne menschliche Intervention und weitgehend selbstverwaltet
- 5 (**Intelligenz**) System als Ganzes beherrscht viele dynamische Aktivitäten und Interaktionen

Mobile Computing

Besonderheiten

- Viele unterschiedliche mobile Geräte: Smartphones, Tablets, GPS Geräte, Fernbedienung etc.
- Mobil impliziert Änderung des Ortes über die Zeit \Rightarrow lokale Dienste, Erreichbarkeit etc. ändern sich – Stichwort **Discovery**.
- Kommunikation ggf. erschwert: keine stabile Route / keine garantierte Verbindung \Rightarrow **Störungstolerante Vernetzung**. 

Sensornetze

Eigenschaften

Die **Knoten** der Sensoren sind:

- Viele (10-1000e)
- Einfach (wenig Speicher-/ Rechen-/ Kommunikationskapazität)
- Oft batteriebetrieben (oder sogar batterielos)