Verteilte Systeme

Einführung

Sommersemester 2026

Christian Berger
Nach Folien von Tobias Distler

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Systemsoftware)

Lehrstuhl für Informatik 4
Systemsoftware

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Überblick

Verteilte Systeme

- Typische Merkmale verteilter Systeme
  - **Mehrere Rechner**, die unabhängig voneinander ausfallen können
  - **Verbindung durch ein Netzwerk**
    - Interaktion nur durch Nachrichtenaustausch möglich
    - Netzwerk unzuverlässig und mit variablen Latenzen
    - Moderate Übertragungsgeschwindigkeit im Vergleich zu Mehrkernsystemen

    $\rightarrow$ Unterschied zu Parallelrechnern
  - **Kooperation der Rechner** zur Lösung einer gemeinsamen Aufgabe
- Definition von [Tanenbaum et al.]

> „Ein verteiltes System ist eine **Kollektion unabhängiger Computer**, die ihren Nutzern wie ein einzelnes **einheitliches System** erscheint.“

- Literatur

Andrew S. Tanenbaum and Maarten van Steen
Distributed systems: Principles and paradigms (2nd edition)
Prentice-Hall, Inc., 2006.

Überblick

Verteiltheit und ihre Konsequenzen

- Ausprägungen
  - Physische Verteiltheit
    - Hardware: **Erhöhte Latenzen** aufgrund von Entfernungen
    - Software: Unabhängige Prozesse auf verschiedenen Rechnern
  - Logische Verteiltheit
    - Aufteilung von Aufgaben auf **mehrere eigenständige Komponenten**
    - Kein Alleinstellungsmerkmal physisch verteilter Systeme
    - Problemstellungen verteilter Systeme auch in Mehrkernrechnern zu finden
- Konsequenzen
  - Knoten haben **nicht zwingend dieselbe Sicht** auf den Systemzustand
  - Unschärfeprinzip: Knotensicht auf den Systemzustand ist
    - entweder aktuell, aber unvollständig
    - oder vollständig, aber veraltet
  - Herangehensweise
    - Kein Verlass auf globale Sichten
    - **Kombination der lokalen Sichten** unterschiedlicher Knoten

Eigenschaften und Herausforderungen

- Nebenläufigkeit
  - Rechner stellen Betriebsmittel zur gemeinsamen Nutzung zur Verfügung
  - „Gleichzeitige“, sich **überlappende Zugriffe sind wahrscheinlich**
  - Koordinierung der Zugriffe erfordert mitunter verteilte Algorithmen

- Fehlerbehandlung
  - Umgang mit **verschiedenen Fehlerklassen**
    - Ausfälle
    - Beliebiges Fehlerverhalten (*Byzantinische* Fehler)
  - Wünschenswerte Maßnahmen sind nicht in jedem Fall praktikabel
    - Fehlererkennung
    - Fehlertolerierung
  - Wiederherstellung nach unterschiedlichen Fehlerarten
    - Transiente Fehler: **Automatisierte Mechanismen** zur Konsistenzwahrung
    - Permanente Fehler: Manuelle Reparatur fehlerhafter Komponenten

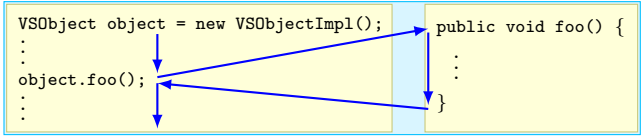
Eigenschaften und Herausforderungen

- Heterogenität
  - Bestandteile eines verteilten Systems sind **nicht notwendigerweise gleichartig**
  - Beispiele
    - Hardware: Smartwatch vs. Server
    - Programmiersprachen: Java vs. C/C++
- Indirekte Bindung
  - Konfigurierung als dynamischer Vorgang
  - **Bindungsunterstützung zur Laufzeit** erforderlich
- Kein einheitlicher Namensraum
  - Zusammenschluss **verschiedener Namensräume**
  - Namensauflösung muss gegebenenfalls über Verwaltungsgrenzen erfolgen
- Kein gemeinsamer Speicher
  - Gültigkeit von Speicherreferenzen ist auf jeweiligen Rechner beschränkt
  - Maßnahmen zum **Umgang mit Referenzen** erforderlich

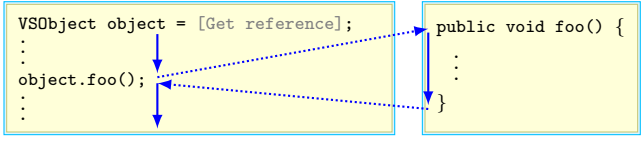
Überblick

Vom lokalen zum verteilten System

- Methodenaufruf im **lokalen System**
  - Ausführung der Methode im Thread des Aufrufers
  - Kein Fortschritt beim Aufrufer während der Methodenausführung



- Methoden**fern**aufruf im **verteilten System**
  - Rollenverteilung: Aufrufer (*Client*) und Aufgerufener (*Server*)
  - Client-Prozess unabhängig vom Server-Prozess
  - Anwendungsbeispiel: Auslagerung aufwändiger Operationen



Transparenz

- Zentrale Frage: **Soll die Verteiltheit dem Client verborgen werden?**
  - Ja $\rightarrow$ Ähnliche Sicht wie im lokalen Fall
  - Nein $\rightarrow$ Verteiltheit als Basis für Entwicklungsentscheidungen

- Netzwerktransparenz
  - Zugriffstransparenz
    - Lokale und entfernte Zugriffe erfolgen über **identische Aufrufe**
    - API: Keine Unterscheidung zwischen lokalen und entfernten Operationen
  - Ortstransparenz
    - Zugriff auf Ressourcen **ohne Wissen über ihren Aufenthaltsort**
    - System muss sich um das Auffinden von Ressourcen kümmern

- Migrations-/Mobilitätstransparenz
  - Ortswechsel von Clients oder Ressourcen zur Laufzeit möglich
  - Keine Beeinträchtigung durch **Ortswechsel von Komponenten**

Replizierte Systeme

- **Redundante Auslegung** der Server-Seite
  - Fehlertoleranz
    - Schutz vor Server-Ausfällen
    - Steigerung der Verfügbarkeit eines Diensts
  - Mehraufwand für Konsistenzwahrung
- Georeplikation
  - Verbesserte Fehlertoleranz durch **geografische Verteilung der Replikate**
  - Leistungseinbußen aufgrund hoher Latenzen
- **Replikationstransparenz**
  - Server-Seite erscheint dem Client als eine Einheit
  - Vorteil: Fehlertransparenz durch dynamischen Replikatwechsel realisierbar
  - Nachteil: Automatische Wahl des Replikats in manchen Fällen ineffizient