



## Verteilte Betriebssysteme

### 2. Kapitel Charakterisierung verteilter Systeme

Prof. Matthias Werner

Professur Betriebssysteme

## Aufgaben von Betriebssystemen

- ▶ Betriebssystem hat i.A. drei Aufgabenbereiche:
  - ▶ Hilfsmittel zur Abarbeitung von Programmen ⇒ **Programmlader**
    - ▶ ...in BS für verteilte Systeme also Hilfsmittel zur Abarbeitung von **verteilten Programmen** und/oder von **Programmen in verteilten Umgebungen**
  - ▶ Abstraktion der konkreten Hardwareeigenschaften und Systemsoftwarekomponenten ⇒ **abstrakte Maschine**
    - ▶ ...in BS für verteilte Systeme also (auch) Abstraktion von **Aspekten der Verteiltheit**
  - ▶ Koordination von Ressourcen ⇒ **Ressourcenmanager**
    - ▶ ...in BS für verteilte Systeme also (auch) **entfernte und verteilte Ressourcen**

## 2.1 Betriebssysteme

### Definition

Jede Software, die die Ausführung von Anwendungen **generisch** unterstützt, kann als Betriebssystem aufgefasst werden.

- ▶ Unterstützung durch virtuelle Ressourcen und Funktionalität
  - ▶ Top-Down-Sicht: Bereitstellung
  - ▶ Bottom-Up-Sicht: Verwaltung
- ▶ Dann **kann** auch Betriebssystem sein:
  - ▶ Teile der Firmware (Bsp.: UEFI)
  - ▶ Virtuelle Laufzeitumgebungen (Bsp.: JVM, .NET CLR, Skript-Interpreter)
  - ▶ Container-Anwendungen (Bsp: Browser, Eclipse)
  - ▶ Middleware-Stacks (Bsp.: Java EE, .NET, CORBA, DCE, ...)

## 2.2 Verteilte Systeme

### Definition nach LESLIE LAMPOR<sup>1</sup>

A **distributed system** is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

- ▶ **Begriffliche Annäherung**  
Ein verteiltes System ist eine Zusammenfassung **autonomer** Rechner, die durch ein Netzwerk verbunden sind und durch geeignete Softwareunterstützung sich dem Benutzer als ein **zusammenhängendes System** darstellen.

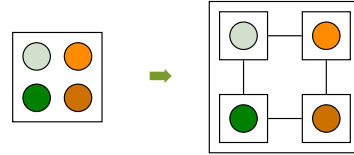
<sup>1</sup>Vgl.: <http://research.microsoft.com/en-us/um/people/lampor/pubs/distributed-system.txt>

## Entstehung: Verteilen vs. Verbinden

Verteilte Systeme können das Resultat zweier Arten von Prozessen sein:

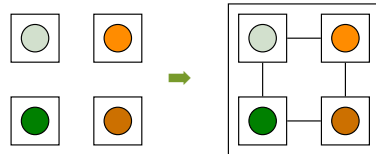
### ► Distribute

Programme und Daten werden auf mehrere Rechner verteilt, um Beschleunigung oder Fehlertoleranz zu erzielen.



### ► Connect

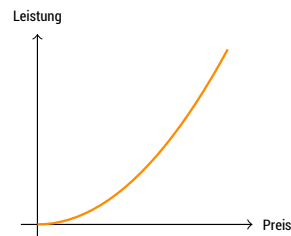
Rechner und ihre Programme werden von anderen Rechnern erreichbar und bilden große Verbunde logischer und physikalischer Betriebsmittel.



## 2.3 Quantitative Ziele

Steigerung der Leistung

Das Gesetz von GROSCH<sup>2</sup> [Herbert R. J. Grosch, 1918 – 2010, US-] „Die Leistung eines Rechners steigt (ungefähr) quadratisch mit dem Preis.“



- **Konsequenz:** Es ist günstiger, einen doppelt so schnellen Rechner zu kaufen als zwei einfach schnelle.
- Das Gesetz galt in den Sechzigern und Siebzigern über einen weiten Bereich von Universalrechnern.

## Quantität und Qualität

- Verteilte Infrastrukturen sind nicht automatisch besser
- Sie werden nur eingesetzt, wenn es **vorteilhaft** ist
- Verteilte Betriebssysteme sollen diese Vorteile unterstützen
- **Quantitative Ziele** lassen sich i.d.R. durch **Leistungseigenschaften** beschreiben
  - 'Verteilen' zur Leistungssteigerung
  - Nutzung zusätzlicher Hardware-Ressourcen
  - Gezielter Vorgang, zeitliche Nähe (Bsp. parallele Anwendung)
- **Qualitative Ziele** lassen sich i.d.R. durch **Transparenzeigenschaften** beschreiben
  - Üblicherweise beim 'Verbinden' von Anwendungskomponenten
  - Verteiltheit ist vorgegeben und soll beherrscht werden

## Steigerung der Leistung (Forts.)

- Aufkommen leistungsstarker Mikroprozessoren
  - Hohe Integrationsdichte (VLSI)
  - Ein-Chip-Prozessoren
  - Automatisierter Herstellungsprozess
  - Automatisierter Entwurfsprozess
  - Hohe Stückzahlen
- **Konsequenz:**
  - Grosch'sches Gesetz gilt nicht mehr
  - 1000 billige Mikroprozessoren bringen (rechnerisch) mehr Leistung (MFLOPS) als teurer Superrechner (z.B. Cray)
- **Idee der 80'er Jahre:**  
Zur Erzielung hoher Leistung bei geringen Kosten viele Prozessoren gemeinsam einsetzen → Parallelverarbeitung („Verteilen“)

## Quantitative Bewertung

- ▶ Es sei
  - $T(1)$  die benötigte Programmbearbeitungszeit auf einem Einprozessorsystem
  - $T(p)$  die benötigte Programmbearbeitungszeit auf einem  $p$ -Prozessorsystem
- ▶ Der Zeitgewinn durch Parallelverarbeitung wird ausgedrückt durch

### Geschwindigkeitsfaktor (Speed-up)

$$S(p) = \frac{T(1)}{T(p)}$$

- ▶ Normiert man den Speed-up auf die Anzahl eingesetzter Prozessoren  $p$ , so entsteht die sogenannte **Effizienz**

### Effizienz

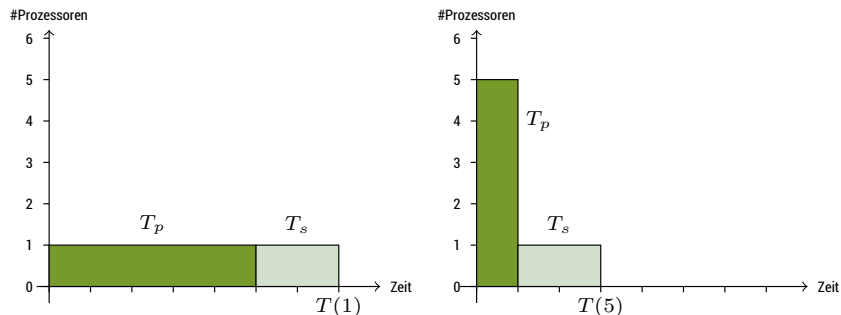
$$E(p) = \frac{S(p)}{p}$$

## AMDAHLS Gesetz (Forts.)

$$T_p = (1 - s) \cdot T(1)$$

$$T_s = s \cdot T(1)$$

### Beispiel:



## AMDAHLS Gesetz

- ▶ Bei Parallelverarbeitung gibt es auch sequentielle Anteile (Bsp: Gesamtergebnis der parallelen Verarbeitung abspeichern)
- ▶ Zerlegung der Bearbeitungszeit in einen sequentiellen und einen parallelen Teil ergibt:

$$T(1) = T_s + T_p$$

- ▶ Es sei

$$s = \frac{T_s}{T_s + T_p}, \quad 0 \leq s \leq 1$$

der sequentielle Anteil eines Programms.

- ▶ Dann gilt das **Gesetz von AMDAHL**:

$$T(p) = s \cdot T(1) + \frac{(1 - s) \cdot T(1)}{p} = T_s + \frac{T_p}{p}$$

## AMDAHLS Gesetz (Forts.)

- ▶ Unter der Annahme von AMDAHL'S Gesetz erhält man für den Speedup mit  $p$  Prozessoren:

$$S(p) = \frac{T(1)}{T(p)} = \frac{p}{1 + s(p - 1)} = \frac{1}{s + \frac{1-s}{p}}$$

- ▶ ... und für die Effizienz

$$E(p) = \frac{S(p)}{p} = \frac{1}{1 + s(p - 1)}$$

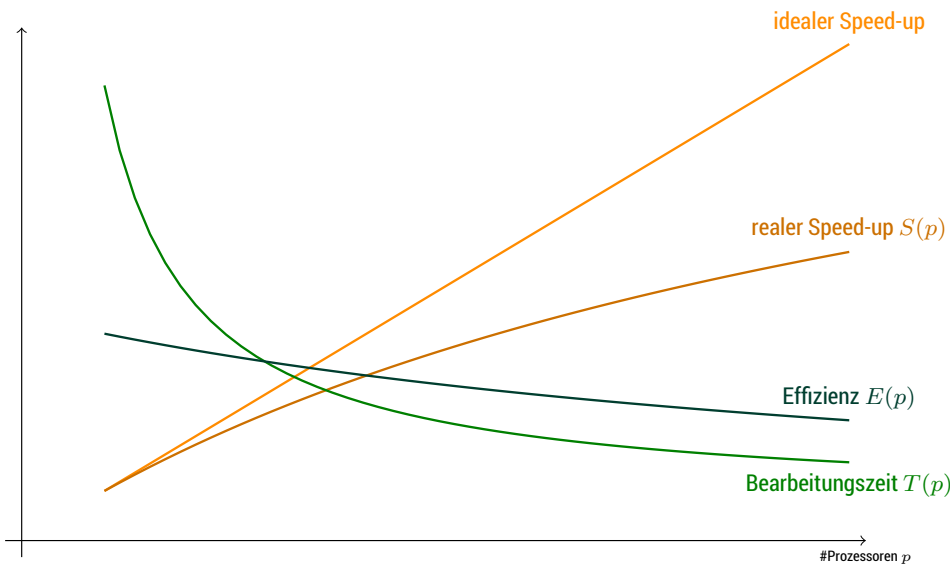
- ▶ Es gilt:

$$\lim_{p \rightarrow \infty} T(p) = s \cdot T(1) = T_s$$

$$\lim_{p \rightarrow \infty} S(p) = \frac{1}{s}$$

$$\lim_{p \rightarrow \infty} E(p) = 0$$

## Verhalten unter AMDAHLS Modell



## GUSTAFSON'S Gesetz

- ▶ Gustafson und Barsis (1988): Anwender sind typischerweise nicht an Ausführungszeit als einziges Optimierungsziel interessiert
- ▶ Begriff des **scaled speed-up** für  $p$  Prozessoren:

$$S_{scaled}(p) = \frac{T_{ser} + p \cdot T_{par}}{T_{ser} + T_{par}}$$

- ▶ Wird gern als Definition für den **Scale-up** Begriff verwendet
- ▶ Linearer Scale-Up wird damit plausibel
- ▶ Es gibt auch andere Modelle (Work-Span, Karp-Flatt Metrik)

## Speed-Up vs. Scale-Up

Dr. Thomas Puzak, IBM, 2007

„Everyone knows Amdahl's Law, but quickly forgets it.“

- ▶ **Speed-up**: Um wie viel schneller kann ein gegebenes Problem durch Verteilung gelöst werden?
  - ▶ **Linearer Speed-up**: n-fache Ressourcenmenge führt zu n-facher Leistungssteigerung
  - ▶ Folgerung: Unter Amdahl sinkt stets die Effizienz
  - ▶ Große verteilte Systeme für Parallelverarbeitung sind nicht effizient
- ▶ **Scale-up** (Problemgrößenfaktor): Um wie viel größer darf das Problem sein, damit es durch Verteilung in **gleicher Zeit** gelöst werden kann?
  - ▶ **Linearer Scale-up**: n-fache Ressourcenmenge erlaubt Bearbeitung der n-fachen Problemgröße in gleichbleibender Zeit
  - ▶ Keine Lösung für Amdahl's Gesetz, sondern andere Zielstellung

## 2.4 Qualitative Ziele

Transparenz – Definitionen

- ▶ **Transparenz** ist die Eigenschaft, dass der Benutzer/Programmierer/Prozess (fast) nichts von der Existenz einer Schicht/ eines Sachverhalts/ eines Mechanismus merkt
- ▶ In diesem Fall von der **Verteiltheit**
- ▶ Dienstleistung des verteilten Betriebssystems an die darüberliegende Anwendung

### Achtung

Der Transparenzbegriff ist nicht ganz intuitiv. Er meint das Gegenteil der Transparenz im täglichen Leben (=Offenheit, Offensichtlichkeit...).

- ▶ Verschiedenste Arten von Definitionen in der Literatur
- ▶ Beispiel: *ODP<sup>3</sup> reference model* [ISO09]

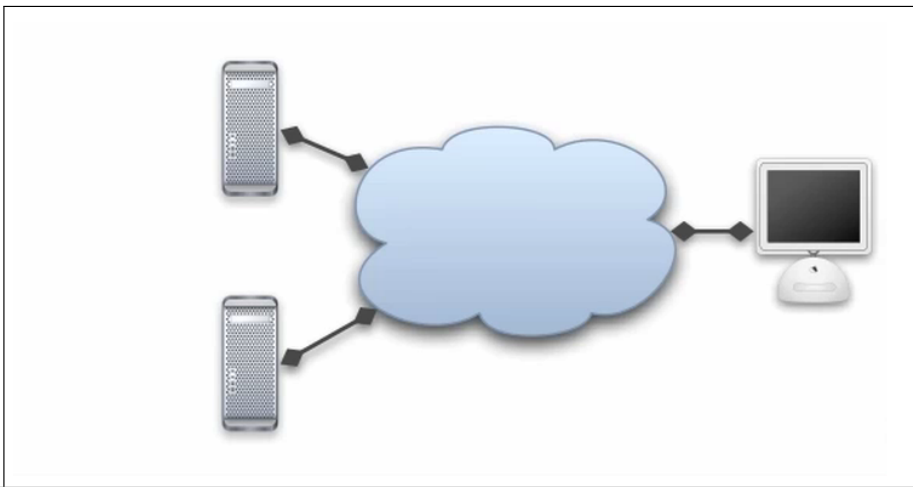
<sup>3</sup>Open distributed processing

## Beispiele

- ▶ **Zugriffstransparenz**  
Zugriffe auf entfernte Ressourcen in gleicher Weise wie auf lokale Ressourcen
- ▶ **Ortstransparenz**
  - ▶ **Namenstransparenz**  
Ressourcen werden über einen Namen angesprochen, der unabhängig von ihrem Standort ist.
  - ▶ **Benutzermobilität**  
Auch wenn der Anwender seinen Standort ändert, kann er auf Ressourcen weiter unter deren Namen zugreifen.
- ▶ **Replikationstransparenz**  
Zur Leistungssteigerung werden automatisch (temporäre) Kopien von Ressourcen verwaltet.
- ▶ **Fehlertransparenz**  
Ausfälle einzelner Komponenten / Knoten sollen möglichst nicht bemerkbar sein.

## Beispiel

- ▶ Transparenz bringt Nutzer und/oder Programmierer Erleichterung
- ▶ Beispiel: Ortstransparenz
  - ▶ Mobile IP (RFC 3344)

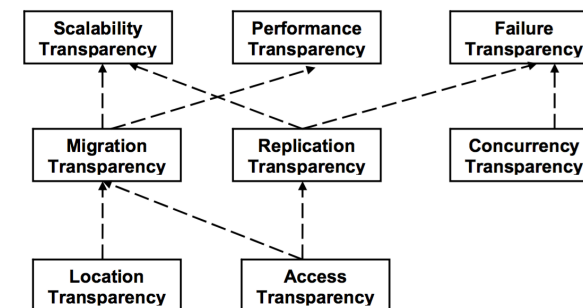


## Beispiele (Forts.)

- ▶ **Migrationstransparenz**  
Die Verlagerung einer Ressource an einen anderen Standort muss von Anwendern nicht berücksichtigt werden.
- ▶ **Nebenläufigkeitstransparenz**  
Die Tatsache, dass mehrere Benutzer auf Ressourcen parallel zugreifen, soll zu keinen Fehlern oder Problemen führen.
- ▶ **Skalierungsstransparenz**  
Die Ausweitung der Anwendung auf weitere Knoten soll im laufenden Betrieb möglich sein. Die Verfahren und Algorithmen sollen das Wachstum „verkräften“ können.
- ▶ **Leistungstransparenz**  
Das verteilte Betriebssystem nutzt die vorhandenen Knoten gleichmäßig ohne explizite Steuerung aus.

## Abhängigkeiten

- ▶ Offensichtlich stehen verschiedene Transparenzarten miteinander in Beziehung



Bildquelle: Kazi Farooqui, Luigi Logrippio und Jan de Meer. „The ISO Reference Model for Open Distributed Processing: An Introduction“. In: *Computer Networks and ISDN Systems* 27.8 (1995), S. 1215–1229. doi: 10.1016/0169-7552(95)00087-N

## 2.5 Herausforderungen

- ▶ Es gibt stark nebenläufige Aktivitäten → **Koordination**
- ▶ Es gibt keine globale Zeit → **Synchronisation**
- ▶ Es gibt keinen gemeinsamen Speicher → **Nachrichtenaustausch**
- ▶ Systeme können sehr groß sein → **Großsystemeffekte**
  - ▶ Quantität wirkt sich auf die Qualität aus
  - ▶ Kompositionseigenschaften hängen von Anzahl der Knoten ab
- ▶ Fehler und Ausfälle sind Normalzustand → **Fehlertoleranz**
- ▶ Die Topologie ändert sich ständig → **Adaptivität**
- ▶ Alle Komponenten sind heterogen → **Standardisierung**
- ▶ Interaktion findet mit Verzögerung statt → **Caching**
- ▶ Bandbreite ist limitiert → **Replikation**

## 2.6 Zusammenfassung

### Verteilung

#### Vorteile

- ▶ Lokale Kontrolle und Verfügbarkeit
- ▶ Maßgeschneiderte Konfiguration
- ▶ Leichte Erweiterbarkeit ('scale-out')
- ▶ Tolerierung partieller Ausfälle
- ▶ Hohe Leistung durch Parallelisierung
- ▶ Herstellerunabhängigkeit
- ▶ Übereinstimmung mit organisatorischen Strukturen




#### Nachteile

- ▶ Hoher Bedienungs- und Wartungsaufwand
- ▶ Probleme durch Heterogenität
- ▶ Schwierigkeit, korrekte verteilte Software zu erstellen
- ▶ Komplexität des Kommunikationssystems
- ▶ Hoher Aufwand bei Dezentralisierung
- ▶ Sicherheitsprobleme
- ▶ Gesamtkosten schwer abzuschätzen

## Trade-Offs

- ▶ Je nach Hauptziel werden ggf. die anderen Ziele vernachlässigt
- ▶ Zugunsten der Leistung wird mitunter auf einige Transparenzen verzichtet
- ▶ Andererseits werden Leistungseinbußen akzeptiert, wenn es eine „bequeme“ Nutzung gibt
  - ▶ Beispiel: Verteilungstransparenz bei Kommunikation
    - ▶ War bis Ende der 80er Jahre sehr beliebt
    - ▶ Brachte zu viel Einbußen bei Skalierbarkeit und Leistung
    - ▶ Komplexe Fehlersuche bei sehr großen Systemen
- ▶ Adaptive Anpassung zur Laufzeit wird immer wichtiger
- ▶ Ideal von maximaler Transparenz **und** Leistungsfähigkeit kaum erreichbar
- ▶ Im Fokus dieser Vorlesung stehen i.d.R. die Transparenzeigenschaften

## Literatur

-  [Sin97] Pradeep K. Sinha. *Distributed Operating Systems. Concepts and Design.* IEEE Press, 1997, Abschnitte 1.5 & 1.6
-  [HVZ96] V. Carl Hamacher, Zvonko G. Vranesic und Safwat G. Zaky. *Computer Organization.* McGraw-Hill, 1996, Abschnitt 10.9
-  [ISO09] ISO/IEC. *Open distributed processing – Reference model.* ISO/IEC 10746-3:2009. 2009