Verteilte Betriebssysteme

2. Kapitel
Charakterisierung verteilter Systeme

Matthias Werner
Professur Betriebssysteme

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, ...)

Aufgaben von Betriebssystemen

- Betriebssystem hat i.A. zwei Aspekte:
  - Abstraktion der konkreten Hardwareeigenschaften und Systemsoftwarekomponenten ➡ **abstrakte Maschine**.
    Hier also auch...
    - Abstraktion von Aspekten der Verteiltheit
    - Hilfsmittel zur Abarbeitung von verteilten Programmen und/oder von Programmen in verteilten Umgebungen
  - Koordination von Ressourcen ➡ **Ressourcenmanager**.
    Hier also auch...
    - Management von entfernten und verteilten Ressourcen

2.2 Verteilte Systeme

Definition nach Leslie Lamport[1]

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.

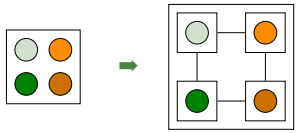
[1] Vgl.: `http://research.microsoft.com/en-us/um/people/lamport/pubs/distributed-system.txt`

Entstehung: Verteilen vs. Verbinden

Verteilte Systeme können das Resultat zweier Arten von Prozessen sein:

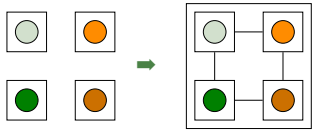
- **Verteilen**

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



- **Verbinden**

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



Quantität und Qualität

- Verteilte Infrastrukturen sind nicht automatisch besser
- Sie werden nur eingesetzt, wenn es **vorteilhaft** ist

- **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

Quantität und Qualität (Forts.)

- Quantitative Ziele sind in der Regel auf Anwendungsregel relevant
  - Wichtigstes quantitatives Ziel ist Rechenperformance
  - Aber auf Verfügbarkeit, Zuverlässigkeit etc. sind quantitative Ziele

- Qualitative Ziele sind vor allem für das Betriebssystem relevant ➡ „schönere“ Maschinen

In dieser Veranstaltung werden wir uns vor allem auf Probleme der Transparenz konzentrieren.

Allerdings gibt es häufig einen Tradeoff zwischen quantitativen und qualitativen Zielen ➡ die Beeinträchtigung sollte nicht überhandnehmen.

Exkurs: Quantitative Ziele und Amdahls Gesetz

- 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 **Speed-up**

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

Geschwindigkeitsfaktor (Speed-up)

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

Effizienz

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

Exkurs: Quantitative Ziele und Amdahls Gesetz (Forts.)

- 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

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

der sequentielle Anteil eines Programms.

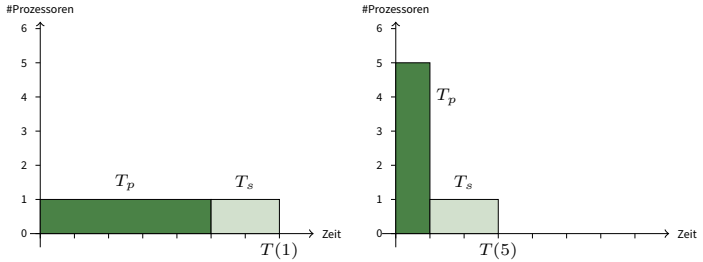
- Dann gilt das **Gesetz von Amdahl**:

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

Exkurs: Quantitative Ziele und Amdahls Gesetz (Forts.)

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

Beispiel:



Exkurs: Quantitative Ziele und Amdahls Gesetz (Forts.)

- Unter der Annahme von Amdahls 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

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

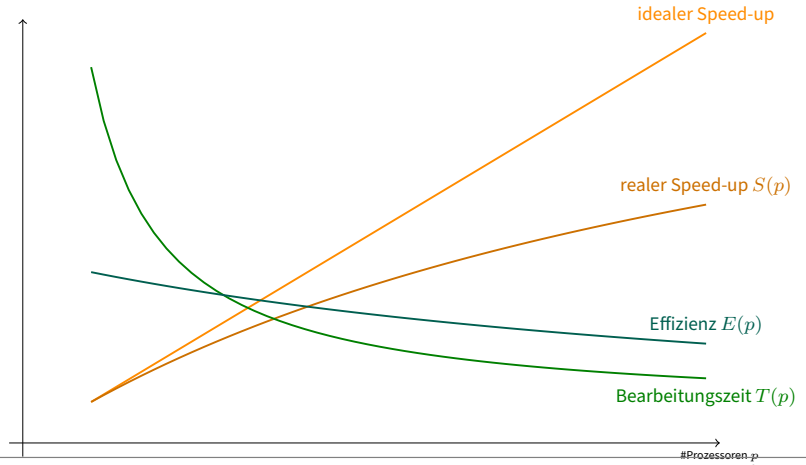
- Es gilt:

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

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

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

Exkurs: Quantitative Ziele und Amdahls Gesetz (Forts.)



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*[2] *reference model* [ISO10746]

[2] Open distributed processing

Beispiele

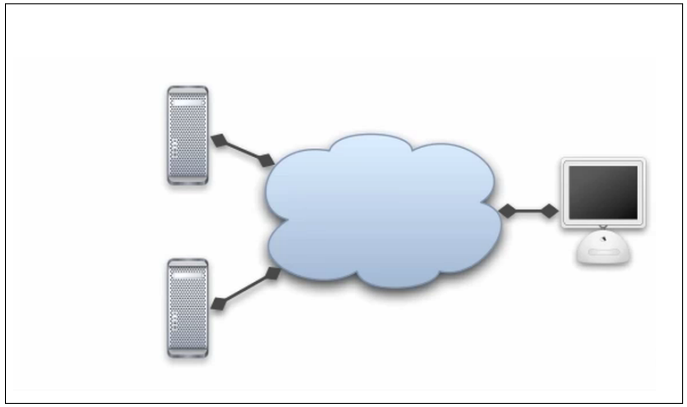
- **Zugriffstransparenz**
  Zugriffe auf entfernte Ressourcen in gleicher Weise wie auf lokale Ressourcen
- **Ortstransparenz**
  - **Namenstransparenz**
    Ressourcen werden über einen Namen angesprochen, der unabhängig von ihrem Standort ist.
  - **Benutzermobilität**
    Auch wenn der Anwender seinen 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.

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 „verkraften“ können.
- **Leistungstransparenz**
  Das verteilte Betriebssystem nutzt die vorhandenen Knoten gleichmäßig ohne explizite Steuerung aus.

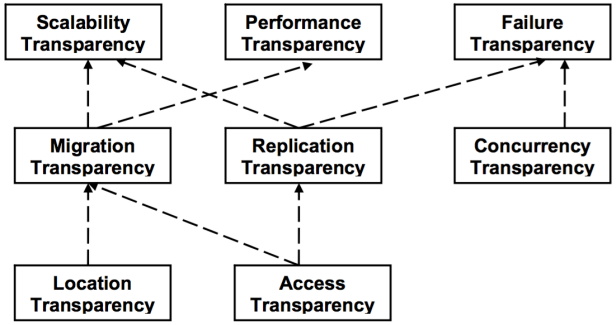
Beispiel

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



Abhängigkeiten

- Offensichtlich stehen verschiedene Transparenzarten miteinander in Beziehung



Bildquelle: Farooqui , K. , L. Logrippo und J. de Meer: „The ISO Reference Model for Open Distributed Processing: An Introduction". *Computer Networks and ISDN Systems*, 27(8)1995, 1215–1229

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**

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

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

Literatur

[Sin97] Sinha , P. K. : *Distributed Operating Systems*. IEEE Press, 1997 , Abschnitte 1.5 & 1.6

[HVZ96] Hamacher , V. C. , Z. G. Vranesic und S. G. Zaky: *Computer Organization*. McGraw-Hill, 1996 , Abschnitt 10.9