Einführung

Verteilte Systeme

Historische Entwicklung

Phase 1: 1930 - 1960 Batch Processing Systeme

- ausführbarer Code inklusive Eingabedaten werden in die Maschine geladen
- Maschine berechnet Ergebnisse und gibt sie aus
- Maschine hält an bzw. nimmt neuen Auftrag an

Phase 2: 1960 - Mitte 80er Timesharing Systeme

- Programme werden in höheren Programmiersprachen geschrieben (Übersetzer/Interpreter)
- Abarbeitung weitgehend sequentiell
- Residentes Betriebssystem ermöglicht Mehrprogramm-betrieb

Historische Entwicklung (2)

Phase 3: Mitte 80er - heute

- Personal Computer und Workstation
- Client/Server-Systeme
- Verteilte Systeme
- Web-Informationssysteme

wird ermöglicht durch technologische Entwicklung:

- mächtige Mikroprozessoren übertreffen Rechenleistung eines Großrechners zum Bruchteil dessen Preises
- Kopplung hin zu Multiprozessoren bietet inkrementelle Leistungsteigerung
- Netzwerke erlauben Kopplung beliebig vieler Rechner

Historische Entwicklung: Netzwerke

PCs/Workstations stellen die Rechnerleistung direkt am Arbeitsplatz zur Verfügung. Diese Insellösungen erlauben keine gemeinsame Nutzung der Betriebsmittel (Drucker, Modem, Daten, Programme, ...).

Lösung:

- verbinde einzelne Rechner durch ein Netzwerk
- jeder Rechner stellt Betriebsmittel zur Verfügung und kann auf andere zugreifen
- alle Rechner sind gleichberechtigt

→ **Peer-To-Peer Netz**

Historische Entwicklung: Netzwerke (2)

In Peer-To-Peer Netzen muss auf jedem Rechner festgelegt werden, wer welche Ressourcen nutzen darf. Hoher Verwaltungsaufwand!

Lösung: zentralisiere Dienste auf bestimmten Rechner

→ **Client/Server-Systeme**

realisiert durch Aufsätze auf bestehende Betriebssysteme:

- ermöglichen den Zugriff auf entfernte Ressourcen
- und organisieren den Server-Betrieb.

→ **Netzwerkbetriebssysteme**

Benutzer wissen von der Existenz anderer Rechner.

Historische Entwicklung: Netzwerke (3)

Erweiterter Ansatz:

- verteile die Betriebssystemfunktionalitäten
- lasse das gesamte Netz wie einen Rechner wirken

→ **Verteilte Betriebssysteme**

- die Netzwerkaspekte des Systems sind vor dem Anwender verborgen (Transparenz)
- der Benutzer sieht nur eine (virtuelle) Maschine
- keine Bindung an dedizierten lokalen Rechner, Anwender kann irgendeinen Rechner des Systems benutzen

Verteilte Systeme: Definitionsversuch

G. Coulouris, J. Dollimore, T. Kindberg:

> Als Verteiltes System wird ein System bezeichnet, bei dem sich die Hardware- und Softwarekomponenten auf vernetzten Rechnern befinden und nur über den Austausch von Nachrichten kommunizieren und ihre Aktionen koordinieren.

Lamport: nicht ganz ernst gemeint ;-)

> Ein verteiltes System ist ein System, mit dem ich nicht arbeiten kann, weil ein Rechner abgestürzt ist, von dem ich nicht einmal weiß, dass es ihn überhaupt gibt.

Verteilte Systeme: Definitionsversuch (2)

A. Tanenbaum, M. van Steen:

> Ein verteiltes System ist eine Menge voneinander unabhängiger Computer, welche dem Benutzer wie ein einzelnes, kohärentes System erscheinen.

Ergänzung:

- die Rechner sind durch ein Netzwerk verbunden
- erscheinen als einzelnes, kohärentes Rechensystem durch geeignete Software-Unterstützung

kohärent: [lat.] zusammenhängend

Grundeigenschaften verteilter Systeme

- Parallele/Nebenläufige Aktivitäten

  → **Koordination und Synchronisation erforderlich**

- Keine globale Uhr und nur begrenzte Synchronisationsmöglichkeiten (durch Signallaufzeiten) für die lokalen Uhren

  → **Timing-Problem**

  Beispiel: Make-Utility, wenn Compiler und Editor auf unterschiedlichen Rechnern ohne globale Uhr laufen.

- Interaktion durch Nachrichtenaustausch (Sockets, aber auch RPC und CORBA)
- Systeme/Anwendungen können sehr groß sein

Grundeigenschaften verteilter Systeme (2)

- Fehler und Ausfälle sind wahrscheinlich. Fehlerquellen: lokaler Rechenknoten, Verbindungsstrukturen, Kommunikation, ...

  → **Fehlertoleranz notwendig**

- Heterogene Hardware- und Softwarekomponenten

  → **Standardisierung von Schnittstellen erforderlich**

- Nachrichtenaustausch über (öffentliche) Netzwerke, Ressourcen gemeinsam nutzen

  → **Sicherheitsprobleme**

Motivation

Erhöhter Nutzwert durch Ressourcenteilung:

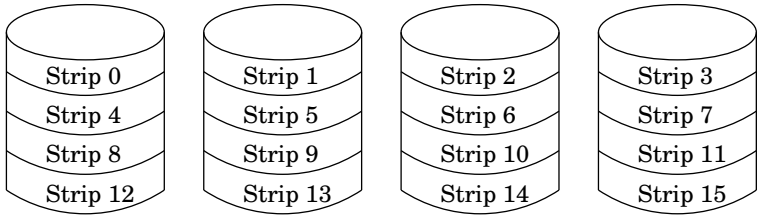
- Ressource: Komponenten, die sich in einem vernetzten Rechnersystem sinnvoll gemeinsam nutzen lassen
- Beispiele: Drucker, Festplatten, Datenbanken, Dateien, Kalender, gemeinsame Dienste wie Webserver, Email

Ausfallsicherheit durch Redundanz:

- Redundante Datenspeicherung auf räumlich verteilten Datenträgern
- Redundante Ausführung von Rechenoperationen fangen einzelne Serverausfälle ab
- Beispiel: RAIDs (Redundant Array of Independant Disks)

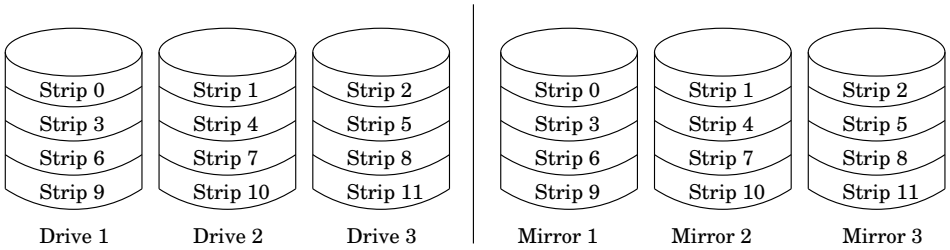
RAID level 0

- fasse Laufwerke zu einem logischen Laufwerk zusammen
- blockweises Verteilen der Daten auf die Laufwerke → Ausfall einer Platte: gesamte Information geht verloren
- Striping-Faktor: Größe der Stücke, die jeweils auf die Laufwerke geschrieben werden
- kleiner Faktor: erhöhte Transferrate, da alle Laufwerke gleichzeitig transferieren, aber Nachteile bei kurzen Requests.



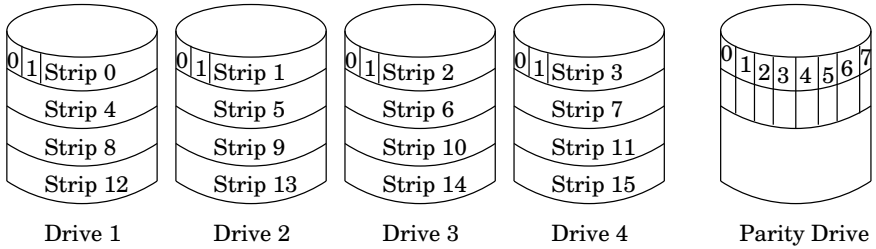
RAID level 1

- spiegeln aller Daten einer Platte/eines RAID-0-Sets auf eine zweite Platte/ein zweites RAID-0-Set
- bei Ausfall einer Platte bleiben die Daten zugreifbar
- Lesezugriff: durch aufteilen auf zwei Laufwerke doppelte Performanz / Schreibzugriff: gleiche Performanz
- hohe Kosten: nur die Hälfte des Plattenplatzes steht für die Originaldaten zur Verfügung



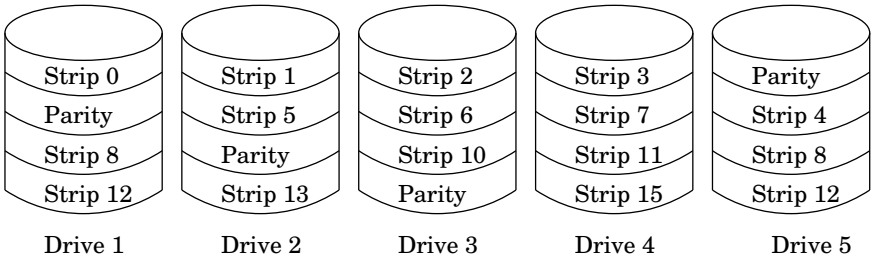
RAID level 4

- zusätzliches Laufwerk mit Parity-Informationen
- Vorteil: bei Ausfall einer Platte können Informationen bit für bit rekonstruiert werden
- Nachteil: geringere Performance als RAID-0/1 durch Lesen und Schreiben zweier Platten
- Flaschenhals: Parity-Laufwerk



RAID level 5

- Parity-Informationen und Daten werden blockbereichsweise auf die Platten verteilt: Jedes Laufwerk ist für einen bestimmten Blockbereich Parity-Laufwerk.
- Performance deutlich geringer als Einzellaufwerk



es gibt weitere RAID-level ...

Software-RAID: Controller ist nicht als Hardware realisiert

Motivation (2)

Beschleunigung der Verarbeitung:

- Einsatz autonomer, vernetzter Verarbeitungseinheiten zur simultanen Verarbeitung unabhängiger Teilprobleme
- Einfachste Vorgehensweise:
  * Aufteilen der Daten in (disjunkte) Teilmengen
  * Verteilen der Teilmengen auf mehrere Prozessoren
  * simultane Verarbeitung und Zusammenfassen der Teilergebnisse zum Endergebnis

Speedup/Effizienz

Lichtgeschwindigkeit ($c_{si} = 3 \cdot 10^{10} mm/s$) setzt prinzipielle Grenze für die Geschwindigkeit bei Einprozessorrechnern: Chip mit 3cm Durchmesser kann Signal in etwas 1ns fortpflanzen $\rightarrow$ 1GFlops

die sinnvolle Anwendung Multiprozessor-/Verteilter Systeme hängt von der dem Problem innewohnenden Parallelität ab: Anzahl gleichzeitig verarbeitbarer Teilaufgaben, in die sich das Problem im Mittel zerlegen lässt.

Speedup/Effizienz (2)

Beispiel: Matrizenmultiplikation

$$(a_{ij}) \cdot (b_{ij}) = (c_{ij} := \sum_{k=1}^{n} a_{ik} \cdot b_{kj})$$

Divide&Conquer: Aufteilen der $n \times n$-Matrizen in jeweils vier $\frac{n}{2} \times \frac{n}{2}$-Matrizen:

$$\left(\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right) = \left(\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right) \cdot \left(\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right) \quad \text{mit} \quad \begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}$$

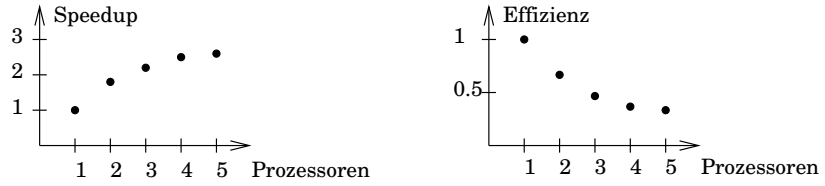
$$C = A \cdot B$$

Die Teilmatrizen r, s, t und u können unabhängig voneinander auf verschiedenen Prozessoren berechnet werden, wenn A und B den Prozessoren zur Verfügung stehen.

Speedup/Effizienz (3)

Leistungsmessung/-bewertung:

- Speedup(n) $= \dfrac{\text{Rechenzeit 1 Prozessor}}{\text{Rechenzeit n Prozessoren}}$
- Effizienz(n) $= \dfrac{\text{Speedup(n)}}{\text{n}}$



Sequentielle und parallele Algorithmen sind oft signifikant verschieden $\rightarrow$ vergleiche besten sequentiellen mit bestem parallelen Algorithmus

Speedup/Effizienz (4)

Amdahls Gesetz: Ein paralleles Programm kann nicht schneller als sein sequentieller Anteil bearbeitet werden, unabhängig von der Anzahl Prozessoren.

Sei f der nur sequentiell ausführbare Programmteil:

$$T_1 = (1 - f) + f = 1$$

Parallelität kann nur den Teil $(1 - f)$ beschleunigen:

$$T_n = (1 - f)/p_n + f$$

Damit ergibt sich für den Speedup:

$$S_n = \frac{1}{(1 - f)/p_n + f} \leq \frac{1}{f}$$

Beispiel verteilter Systeme: Internet

- Zusammenschluss verschiedener Netzwerke / Rechner
- Kommunikation durch Austausch von Nachrichten
- Ortsunabhängige Nutzung von Diensten
  * Bereitstellung von Daten (Text, Bilder, Audio, ...)
  * Diensten (WWW, E-commerce, E-mail, ...)
  * Anwendungen (ASP = Application Service Provider)
- Ortsunabhängiger Zugang zu Rechenressourcen
  * Internet als große Rechenressource bei Projekten
  * Beispiele: GRID-Computing, SETI@home (Search for ExtraTerrestrial Intelligence at home)

Beispiel verteilter Systeme: Intranet

- Teil des Internets, wird separat verwaltet, verfügt dank einer Abgrenzung über lokale Sicherheitsstrategien
- Konfiguration unterliegt der Verantwortlichkeit des Unternehmens
- Ausdehnung nicht lokal beschränkt (mehrere LANs im Intranet möglich) → virtual private networks
- durch besondere Absicherung viele Möglichkeiten zur Ressourcenteilung
- Firewalls: Filterung der ein-/ausgehenden Nachrichten

Aber: Kenntnis über Ort einer Ressource/eines Dienstes notwendig und Zugriffsoperationen sind für lokale/entfernte Ressourcen unterschiedlich

Aufgabenstellungen

Verteilte Systeme basieren auf unterschiedlichen Netzwerken, Hardware, Betriebssystemen, Programmiersprachen, Implementierungen, ...

→ **Heterogenität**

- gemeinsame/standardisierte Netzwerkprotokolle
- austauschbare, hardware-unabhängige Formate für Daten/Datenstrukturen
- Standards für Interprozesskommunikation

Ansätze zur Lösung des Problems:

- erzeuge Code für virtuelle Maschine, nicht für Hardware
- Middleware: Software-Schicht verbirgt Heterogenität

Aufgabenstellungen (2)

→ **Offenheit/Erweiterbarkeit des Systems**

- Grad, zu dem neue Dienste hinzugefügt und zur Verwendung von unterschiedlichen Clients bereitgestellt werden
- Veröffentlichung von Spezifikation/Dokumentation zu Schnittstellen und Internet-Protokolle:
  * RFC-Dokumente (Requests For Comment) enthalten Diskussionen und Spezifikationen
  * Beispiel: für CORBA existiert eine Reihe technischer Dokumente (siehe `http://www.omg.org`)
  * oft wird so der langsame, offizielle Standardisierungsweg umgangen
- herstellerunabhängig

Aufgabenstellungen (3)

→ **Sicherheit**

- Vertraulichkeit: keine Offenlegung der Ressourcen gegenüber nicht-berechtigten Personen/Agenten
- Integrität: Schutz der Ressourcen gegen Veränderung oder Beschädigung
- Verfügbarkeit: Schutz gegen Störungen der Methoden für Ressourcenzugriff wie

  Denial-Of-Service-Angriffe: Einen Server mit vielen sinnlosen Anfragen überfluten, so dass ernsthafte Anfragen nicht mehr beantwortet werden können

  Computerviren: Wie kann automatisch festgestellt werden, ob das mitgelieferte Skript einen Virus enthält? (Sicherheit mobilen Codes)

Aufgabenstellungen (4)

→ **Skalierbarkeit**

- Verteilte Systeme müssen auch bei steigender Anzahl von Benutzern/Komponenten effizient/effektiv arbeiten
- Beispiel: Internet

Datum	Computer	Webserver	Prozentsatz
Ende 1979	188	0	0
Juli 1989	130.000	0	0
Juli 1993	1.776.000	130	0,008
Juli 1997	19.540.000	1.203.096	6
Juli 1999	56.318.000	6.598.697	12

Aufgabenstellungen (5)

Kernprobleme beim Entwurf skalierbarer verteilter Systeme

- Kostenkontrolle der physischen Ressourcen: Bei steigender Nachfrage Erweiterung zu vernünftigen Kosten
- Vermeiden von Leistungsengpässen: bevorzuge hierarchische Lösungen, so dass die Menge zu ladender/verarbeitender Daten nicht zu groß wird (z.B. DNS)
- Erschöpfung der SW-Ressourcen verhindern: Ressourcen so anlegen, dass sie für zukünftige Erweiterungen voraussichtlich ausreichen werden.
  aktuell: Umstellung von 32-Bit Internetadressen auf 128-Bit (Wer hätte vor 20 Jahren gedacht, dass auch Waschmaschinen eine eigene IP-Adresse haben werden?)

Aufgabenstellungen (6)

→ **Nebenläufigkeit**

- zur Erhöhung des Durchsatzes
- macht Sicherungsmechanismen bei parallelen Zugriffen notwendig, z.B. Transaktionskonzept in verteilten Systemen

→ **Fehlerverarbeitung**

- in der Regel schwierig: oft treten partielle Ausfälle auf
- Fehler erkennen: in einigen Fällen problemlos (z.B. Prüfsummen), viele Fehler werden nur vermutet
- Fehler maskieren: Erkannte Fehler können durch Wiederholung der Aktion/Redundanz verborgen oder abgeschwächt werden

Aufgabenstellungen (7)

→ **Fehlerverarbeitung** (Fortsetzung)

- Fehler tolerieren: Fehler anzeigen, Benutzer weitere Entscheidung überlassen, z.B. „Web-Site nicht erreichbar“
- Wiederherstellung nach Fehlern (Journaling File System, Database Recovery, ...)
- Erhöhte Verfügbarkeit/Fehlertoleranz durch Redundanz. Beispiele:
  * Zwei mögliche Routen zwischen Routern im Internet
  * Jede DNS-Tabelle (Domain Name Service) wird auf zwei Hosts repliziert
  * Replikation der Datenbank-Partitionen auf unabhängigen Rechnern oder mittels RAID

Aufgabenstellungen (8)

→ **Transparenz**

Verbergen der räumlichen Trennung der einzelnen Komponenten im verteilten System vor Benutzern/Anwendungen

- Zugriffstransparenz: identische Zugriffsoperationen für lokale und entfernte Ressourcen
- Ortstransparenz: keine Kenntnis über Ort einer Ressource/Dienstes notwendig
- Replikationstransparenz: Ressourcenreplikation zur Verbesserung der Leistung/Zuverlässigkeit sind für Benutzer/Anwendungen unsichtbar
- Mobilitätstransparenz: Verschiebung von Ressourcen/ Clients innerhalb des Systems ohne Beeinträchtigung der Arbeit möglich (Aspekt der Ortstransparenz)

Aufgabenstellungen (9)

→ **Transparenz** (Fortsetzung)

- Leistungstransparenz: dynamische Rekonfiguration bei variierender Last ist möglich (eigentlich Aspekt der Ortstransparenz)

  Idee: Rechenleistung wird vom Netz bereitgestellt; welche Rechner die Leistung erbringen, ist unsichtbar.
- Skalierungstransparenz: Vergrößerung des Systems ohne Veränderung der Systemstruktur und Anwendungen möglich

Vorteile verteilter Systeme

- Kostenreduktion
- Lokale Kontrolle und Verfügbarkeit
- Leichte Erweiterbarkeit
- Ausfalltoleranz
- Hohe Leistung durch Parallelität
- Modulare Software
- Herstellerunabhängigkeit
- Übereinstimmung mit organisatorischen Strukturen

Nachteile verteilter Systeme

- Hoher Bedienungs- und Wartungsaufwand
- Probleme durch Heterogenität
- Komplexer Design- und Implementierungsprozess
- Schwierige Verifikation der Korrektheit
- Komplexe Kommunikationssysteme
- Hoher Aufwand beim Übergang vom zentralen zum dezentralen System
- Sicherheitsprobleme
- Gesamtkosten schwer abschätzbar

Schichtenmodelle

Verteilte Anwendung besteht aus Komponenten:

- graphische Präsentation / Benutzungsoberfläche
  Beispiel: der X-Client übermittelt Tastaturanschläge und Mausbewegungen an den X-Server
- Benutzungsinterface / Graphical User Interface (GUI)
  Beispiele: Eingabemaske, Suchdialog, ...
- Verarbeitung / Anwendung (Applikationslogik)
- Datenmanagement und
- persistente Datenspeicherung (Datenbank, XML-Datei)

→ Komponenten werden auf Rechenressourcen aufgeteilt.

Schichtenmodelle (2)

Bei klassischen Client/Server-Architekturen ergeben sich zwei Stufen: **two tier model**

Klassifikation von Clients: Unterscheidung danach, wo die Trennung zwischen Client und Server gelegt wird.

- **null client** Trennlinie zwischen graphischer Benutzungsoberfläche und -interface: Client übergibt nur Tastaturanschläge und Mausbewegungen an den Server. → host based computing.
- **thin client** Trennlinie zwischen Benutzungsinterface und Anwendung. Client verwaltet GUI und einige Zustandsinformationen, Server verwaltet Applikationslogik und Daten. → remote presentation. Beispiel: NetPC

Schichtenmodelle (3)

- **applet client** Trennlinie liegt hinter Benutzungsinterface und schließt Teile der Applikationslogik ein.
  → cooperative processing
- **fat client** Trennlinie liegt zwischen Applikationslogik und Datenmanagement

Heute: Anwendung zerlegen in einen Client und mehrere Server → mehrschichtige Anwendungen (**multi tier model**)

- bessere Skalierbarkeit bei Servern
- Einschluss der Intranet/Internet-Technologie

Schichtenmodelle (4)

Typisch: dreischichtige Gliederung (**three tier model**)

- Clients für Präsentationsschicht
- Applikations-Server
- Datenbank-Server

Anforderung an verteilte Systeme:
Verteilung der Software-Komponenten ist für den Benutzer transparent. Anwendung soll genauso ablaufen, als ob sie auf einem einzigen Rechner abläuft. → Middleware Services

Die Verteilung der Komponenten erfolgt dabei nicht durch das Betriebssystem, sondern durch Konfiguration.

Middleware

Middleware Services:

- Verbirgt Heterogenität eines VS
- Stellt Entwicklern ein praktisches Programmiermodell zur Verfügung
- Bestandteile: Prozesse und Objekte in mehreren Einheiten des VS, die durch Zusammenarbeit die Kommunikation und gemeinsame Ressourcennutzung unterstützen

Ein Middleware-Service ist verteilt:

- erlaubt entfernten Zugriff (z.B. Datenbankzugriff) oder
- befähigt andere Services zu einem entfernten Zugriff

Middleware (2)

Middleware abstrahiert von entfernten, über das Netz ablaufenden Interaktionen:

- Client benutzt API (Application Programming Interface) der Middleware, die die Anfrage über das Netz zum Server transportiert und den Rücktransport des Ergebnisses des Servers zum Client vornimmt.
- Middleware läuft bei einer Interaktion auf beiden Seiten, dem Client und dem Server
- Middleware unterstützt Standard-Protokoll oder zumindest ein veröffentlichtes Protokoll, z.B. TCP/IP oder IPX (Novell)

Middleware (3)

Middleware stellt Bausteine für den Aufbau von Software-Komponenten bereit, die in einem verteilten System zusammenarbeiten können.

Beispiele solcher Dienste:

- Abstraktion von Kommunikationsaktivitäten: Entfernte Prozedur-/Funktionsaufrufe, Gruppenkommunikation, Ereignisbenachrichtigung, Datenreplikation
- Qualitätssicherung, z.B. Echtzeitübertragung multimedialer Daten
- Dienste für Anwendungen: Namensdienste, Sicherheit, Transaktionen, persistenter Speicher, Zeitdienst, ...

Middleware (4)

Beispiele für Middleware-Produkte bzw. -Standards

- Sun RPC: entfernter Prozeduraufruf
- DCE (Distributed Computing Environment): Dienste und Werkzeuge für verteilte Datenverarbeitung
- Corba (Common Object Request Broker Architecture)
- Java RMI (Remote Method Invocation)
- DCOM (Distributed Common Object Model)

Zusammenfassend: Middleware ist eine Software-Schicht, die auf Basis standardisierter Schnittstellen und Protokolle Dienste für eine transparente Kommunikation verteilter Anwendungen in einem heterogenen Umfeld bereitstellt.

Distributed Computing Environment

Definiert durch OSF (Open Software Foundation)
siehe `http://www.rz.uni-karlsruhe.de/dienste/3219.php`

Primäres Ziel: Herstellerunabhängigkeit

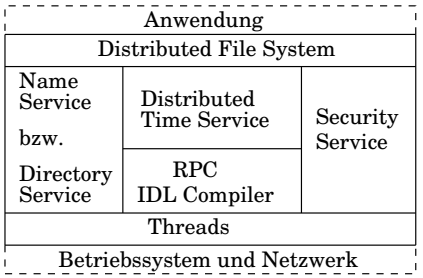
- Aufsatz für Betriebssysteme wie TRU64, AIX, HP-UX, OS/2, Unix, VMS, Windows, ...
- Unterstützung diverser Protokolle wie TCP/IP, X.25
- Unterschiede der Maschinen werden durch automatische Konvertierungen vor dem Anwender versteckt
  → vereinfachte Anwendungsentwicklung

DCE: kein Betriebssystem oder Anwendung, sondern eine Sammlung von Services und Werkzeugen (Middleware)

DCE: Wichtigste Komponenten

- Thread-Package (evtl. im Betriebssystem integriert)
- RPC-Facility: Werkzeuge zur Gestaltung von Prozeduraufrufen auf entfernten Rechnern
- Distributed Time Services: Synchronisation -so weit es geht- der Uhren aller beteiligten Einheiten
- Namensdienst/Directory Service: Ortstransparenter Zugriff auf Server, Dateien, Geräte, ...
- Sicherheitsdienste: Authentifikation und Autorisierung
- Distributed File Services (DFS): Verteiltes Dateisystem
  → einheitlicher, transparenter, systemweiter Zugriff auf Dateien (Vorteile bzgl. Sicherheit, Performance, feinere Rechtevergabe, Verfügbarkeit, ...)

DCE: Grundsätzlicher Aufbau



Jede Komponente nutzt die unter bzw. neben ihr liegenden Komponenten

DCE-System kann tausende von Computern umfassen, die weltweit verteilt sind.

Strukturierung und Gliederung durch Konzept der Zellen. Zelle: · Administrative Einheit von Benutzern, Maschinen und Diensten, die einen gemeinsamen Zweck haben und sich DCE-Services teilen · weltweit eindeutiger Name · ein Rechner kann nur einer DCE-Zelle angehören

Minimale Zellenkonfiguration: Cell Directory Service, Security Service, verteilter Time Service und Client-Maschinen