Technische Hochschule Ingolstadt

Fakultät für Elektrotechnik und Informatik

Zukunft in Bewegung

Die Physische Sicht

Architektur- und Entwurfsmuster der Softwaretechnik

Prof. Dr. Bernd Hafenrichter 14.05.2018

Motivation

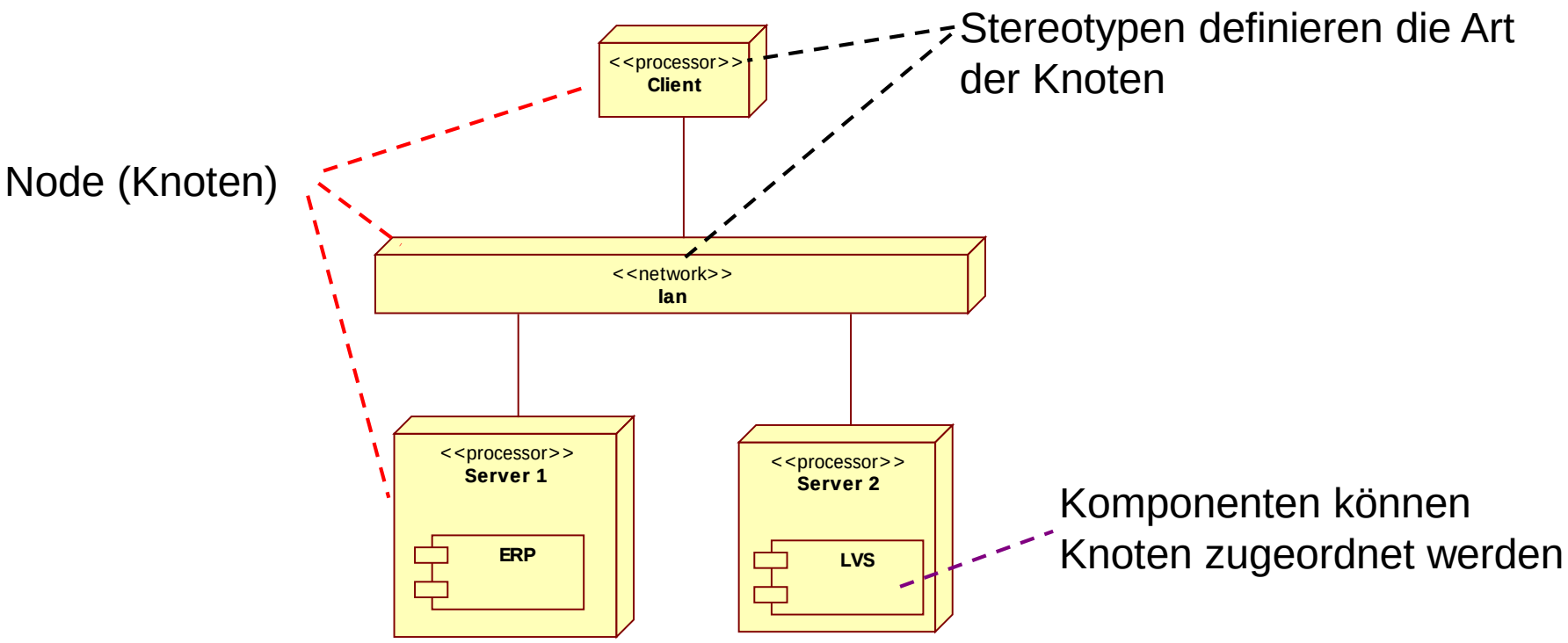
- Physikalische Sicht der Architektur
  - Fokus:
    - Zuordnung der Software auf die physische Hardware sowie Verteilung (= Distributed System)
    - Wie kommunizieren die Komponenten untereinander
  - Sicherstellung der nichtfunktionalen Anforderungen
    - Availability
    - Reliability (fault-tolerance)
    - Performance (throughput)
    - Scalability
    - Security

Motivation

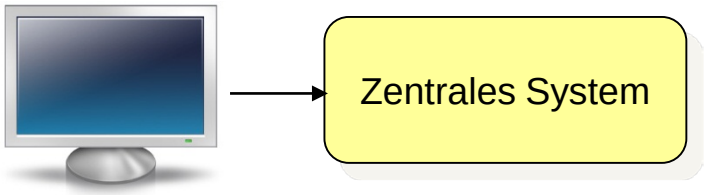
- Für die Aufteilung eines Systems auf verschiedene physische Knoten existieren Verteilungsmuster welche man als Referenz verwenden kann
  - Zentrales System
  - Two-Tier (Thin-Client, Fat-Client)
  - Three-Tier (GUI; Applikationskern, Datenhaltung)
  - Föderation
  - Cluster
  - Service-Oriented-Architecture
  - Microservices

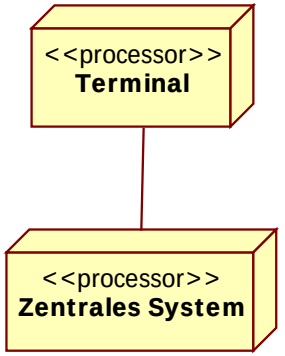
Motivation

- UML-Deploymentdiagramme können für die Dokumentation der Physischen Struktur verwendet werden



Verteilungsmuster "Zentrales System"

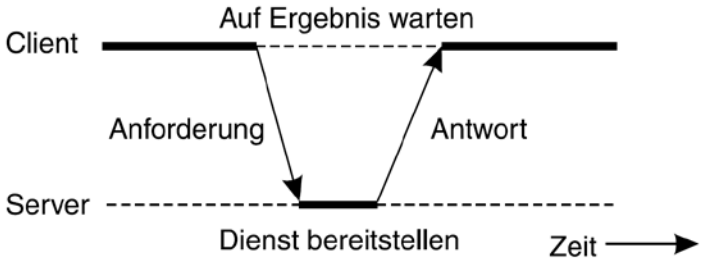


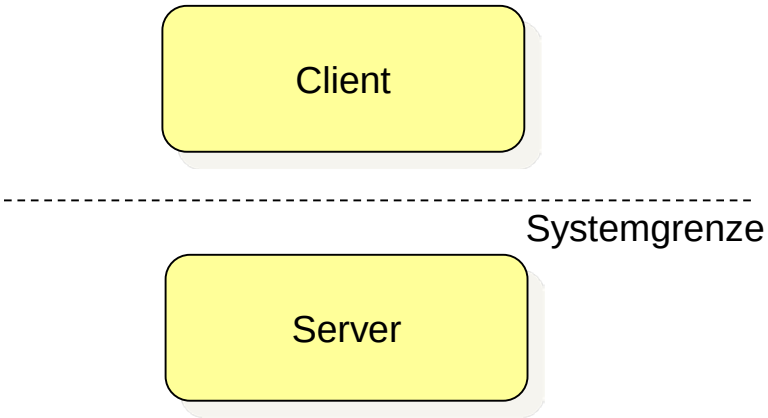


- Beispiele:
  - Klassische Großrechner-("Mainframe"-)Anwendungen
- Noch einfachere Variante:
  - Lokale PC-Anwendungen (Zentrale System und Terminal in einem)

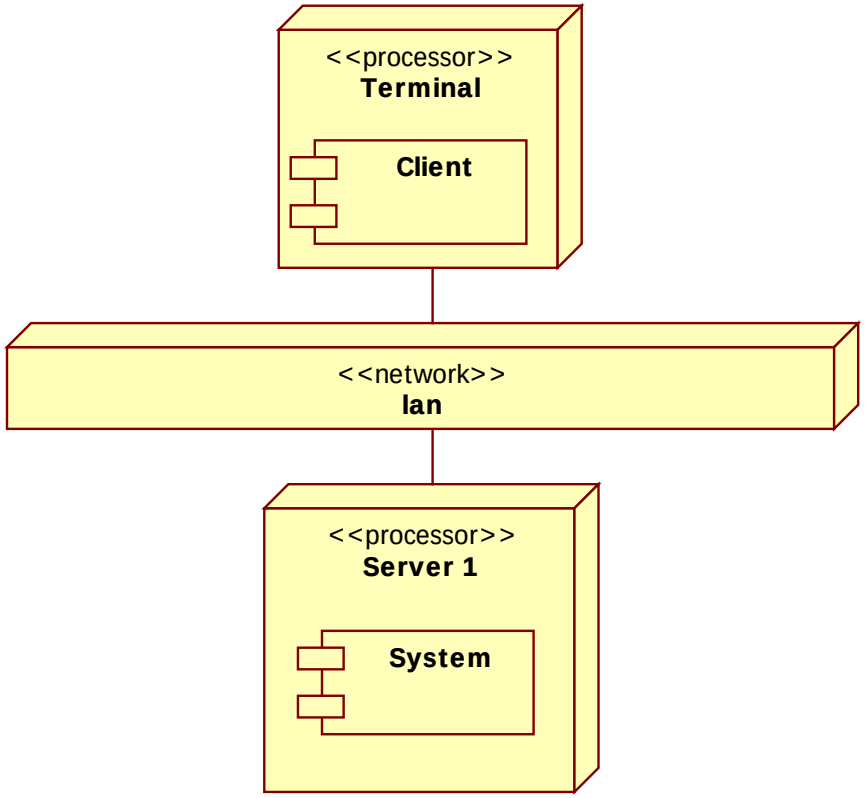
Verteilungsmuster „Client/Server"

- Client-Server ist das traditionell benutzte Modell für verteilte Systeme
- Ein Server stellt standardisierte Dienste zu Verfügung
- Ein Client konsumiert/benutzt einen Dienst
- Dadurch: Reduzierte Komplexität, Bessere Strukturierung des Gesamtsystems
- Clients und Server können sich überlappen, d. h., ein Rechner kann beide Rollen übernehmen
- Die Zusammenarbeit wird als *Anforderungs-Antwort-Verhalten (Request-Response)* bezeichnet





Verteilungsmuster „Client/Server"



Verteilungsmuster „Client/Server"

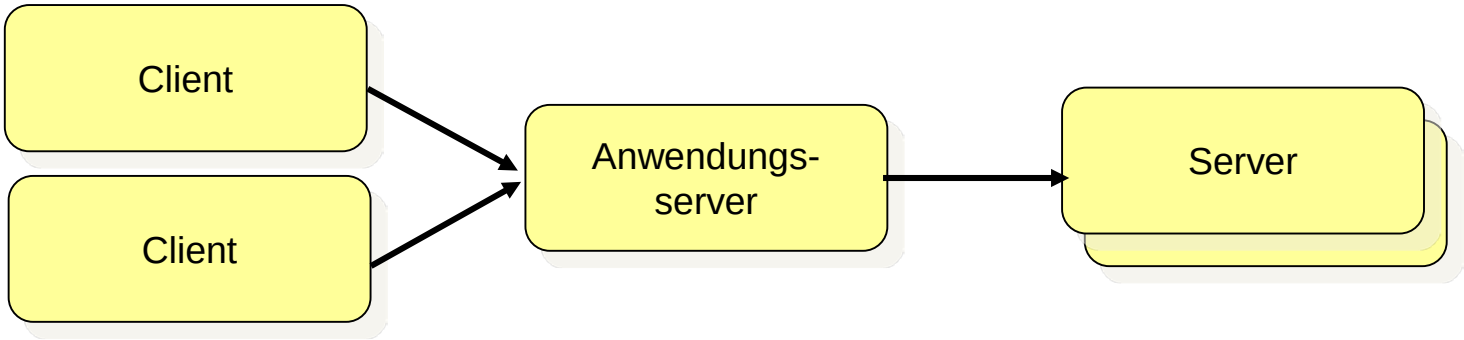
Thin-Client

- Nur die Benutzungsschnittstelle auf dem Client-System
- Keine Anwendungslogik
- i.d.R. Realisiert durch einen WebBrowser der die Rolle des Thin-Client übernimmt

Fat-Client

- Teile der Fachlogik (oder gesamte Fachlogik) auf dem Client-System
- Hauptfunktion des Servers: Datenhaltung
- Entlastung des Servers
- Zusätzliche Anforderungen an Clients (z.B. Installation von Software)

Verteilungsmuster "Three-Tier Client/Server"



Client

- Benutzungsschnittstelle
- evtl. Fachlogik

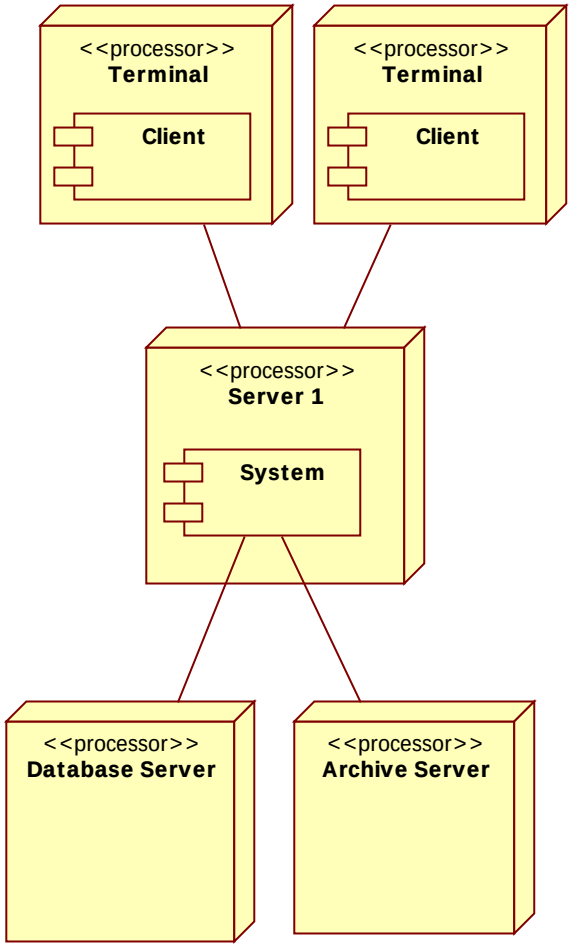
Anwendungsserver

- evtl. Fachlogik
- Verteilung von Anfragen auf verschiedene Server

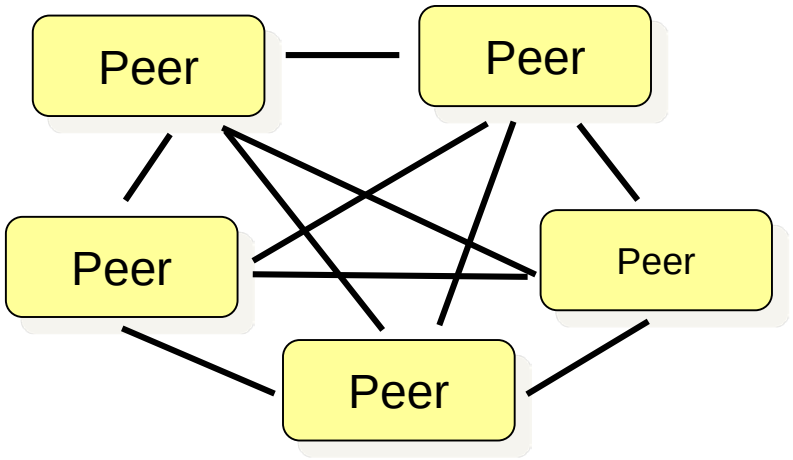
Server

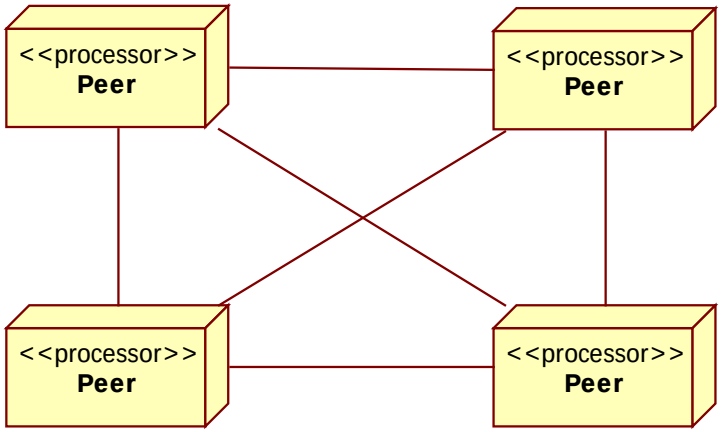
- Datenhaltung, Rechenleistung etc.

Verteilungsmuster „Client/Server"



Verteilungsmuster "Föderation"





- Gleichberechtigte Partner *(peer-to-peer)*
- Unabhängigkeit von der Lokation und Plattform von Funktionen
- Verteilte kommunizierende Objekte

Hochverfügbarkeit

Motivation

- Unternehmensapplikation werden immer kritischer.
- Der Ausfall einer Applikation kann erheblichen Schaden für die Firma nach sich ziehen.
- Die Forderung nach hoch verfügbaren Lösung ist immer wichtiger
- Software-Systeme und Hardware muss so entworfen werden dass die Forderung nach Hochverfügbarkeit realisiert werden kann.

Die folgenden Ausführungen basieren auf: Andrea Held, https://www.informatik-aktuell.de/betrieb/verfuegbarkeit/hochverfuegbarkeit-und-downtime-eine-einfuehrung.html

Hochverfügbarkeit

Kosten pro h Ausfall

Geschäftsfeld	Kosten je Stunde Ausfallzeit
Airline Reservation Centers	67.000$ – 112.000$
Bank ATM Service Fees	12.000$ – 17-000$
Brokerage House (Stock)	5.600.000$ – 7.300.000 $
Catalog Sales Centers	60.000$ – 120.000$
Package Shipping	28.000$ – 32.000$
Credit Card / Sales Authorization	2.200.000$ – 3.100.000$
Pay-per-View Television	67.000$ – 112.000$
Teleticket Sales	56.000$ – 82.000$

Quelle Contingency Planning Research, 2001

Hochverfügbarkeit

Kennzahlen der Systemverfügbarkeit:

- Recovery Point Objective (RPO)
  - Wie hoch darf der maximale Datenverlust sein?
- Recovery Time Objective (RTO)
  - Wie lange darf ein IT-System maximal ausfallen?

Messgrößen zu Beurteilung der Systemverfügbarkeit:

- Mean Time Between Failure (MTBF)
  - Mittlere ausfallfreie Zeit eines Systems

- Mean Time to Repair (MTTR):
  - Mittlere Dauer für die Wiederherstellung nach einem Ausfall

Verfügbarkeit = MTBF / MTBF + MTTR
Nicht Verfügbarkeit = MTTR / MTBF + MTTR

Hochverfügbarkeit

Klassen der Hochverfügbarkeit (Harvard Research Group (HRG)

- **Conventional (AEC-0):** Funktion kann unterbrochen werden, Datenintegrität ist nicht essentiell
- **Highly Reliable (AEC-1):** Funktion kann unterbrochen werden, Datenintegrität muss jedoch gewährleistet sein
- **High Availability (AEC-2):** Funktion darf nur innerhalb festgelegter Zeiten bzw. zur Hauptbetriebszeit minimal unterbrochen werden
- **Fault Resilient (AEC-3):** Funktion muss innerhalb festgelegter Zeiten bzw. während der Hauptbetriebszeit ununterbrochen aufrechterhalten werden
- **Fault Tolerant (AEC-4):** Funktion muss ununterbrochen aufrechterhalten werden, 24*7 Betrieb (24 Stunden, 7 Tage die Woche) muss gewährleistet sein
- **Disaster Tolerant (AEC-5):** Funktion muss unter allen Umständen verfügbar sein

Hochverfügbarkeit

Klassen der Hochverfügbarkeit (Harvard Research Group (HRG)

Verfügbarkeits-Klasse	Bezeichnung	Verfügbarkeit in Prozent	Downtime pro Jahr
2	Stabil	99,0	3,7 Tage
3	Verfügbar	99,9	8,8 Stunden
4	Hochverfügbar	99,99	52,2 Minuten
5	Fehlerunempfindlich	99,999	5,3 Minuten
6	Fehlertolerant	99,9999	32 Sekunden
7	Fehlerresistent	99,999999	3 Sekunden

Hochverfügbarkeit und Downtime, Andrea Held

Hochverfügbarkeit

Downtime: Was führt zu dem Ausfall eines Systems

geplante Downtime:

ein Zeitraum, in dem das System aufgrund geplanter Tätigkeiten nicht verfügbar ist.

- Hardware Upgrades
- Software Upgrade
- Wartungsarbeiten

Hochverfügbarkeit

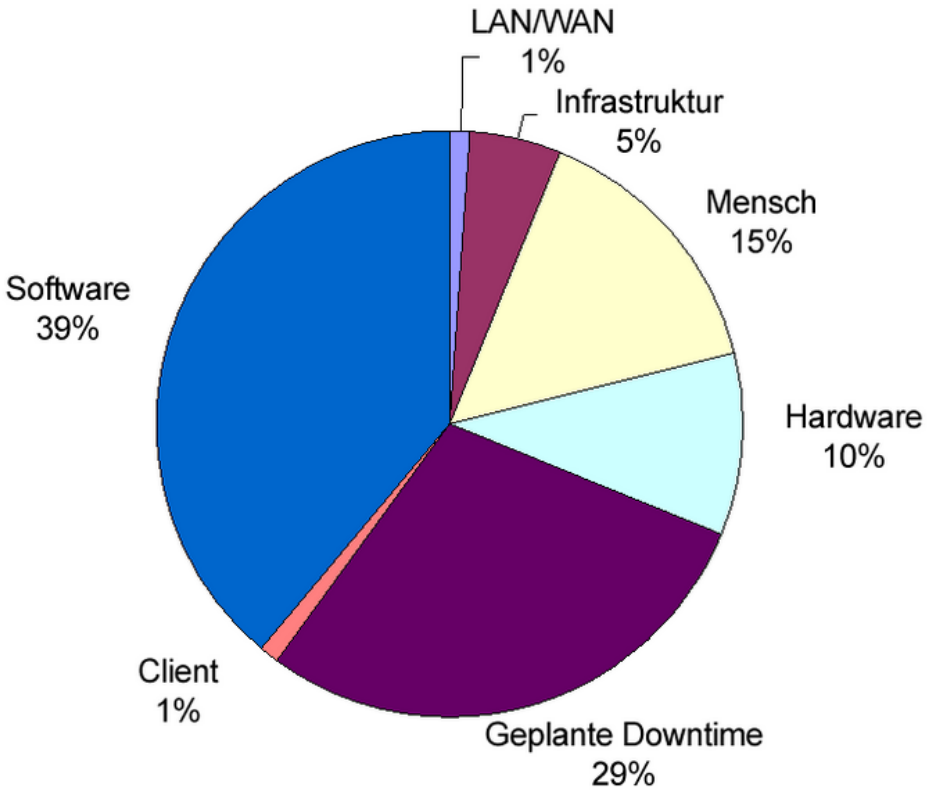
Downtime: Was führt zu dem Ausfall eines Systems

ungeplante Downtime:
eine Zeitspanne bezeichnet, in der das System aufgrund ungeplanter Ereignisse nicht verfügbar ist.

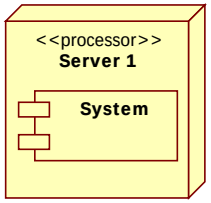
- Defekte Hardware
- Fehlerhafte Software
- Bedienungsfehler
- Stromausfälle
- Netzwerkprobleme:
- Katastrophen / Desaster (Überflutung, Erdbeben, Bombenanschläge)

Hochverfügbarkeit

Prozentuale Aufteilung der Downtime: Was führt zu dem Ausfall eines Systems (IEEE Computer)

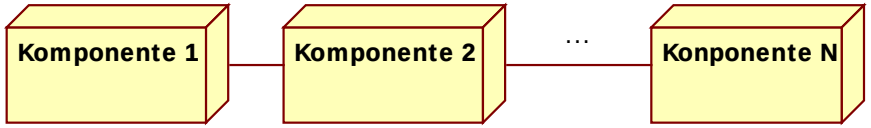


Verfügbarkeit berechnen



Einfachsystem ohne Redundanz

Verfügbarkeit (in %) = A

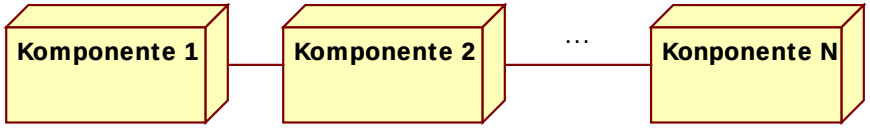


Mehrfachsystem ohne Redundanz

Verfügbarkeit (in %) = $A_1 * A_2 * A_3 \ldots * A_N$

Der Ausfall einer Komponente führt zum Ausfall des gesamten System

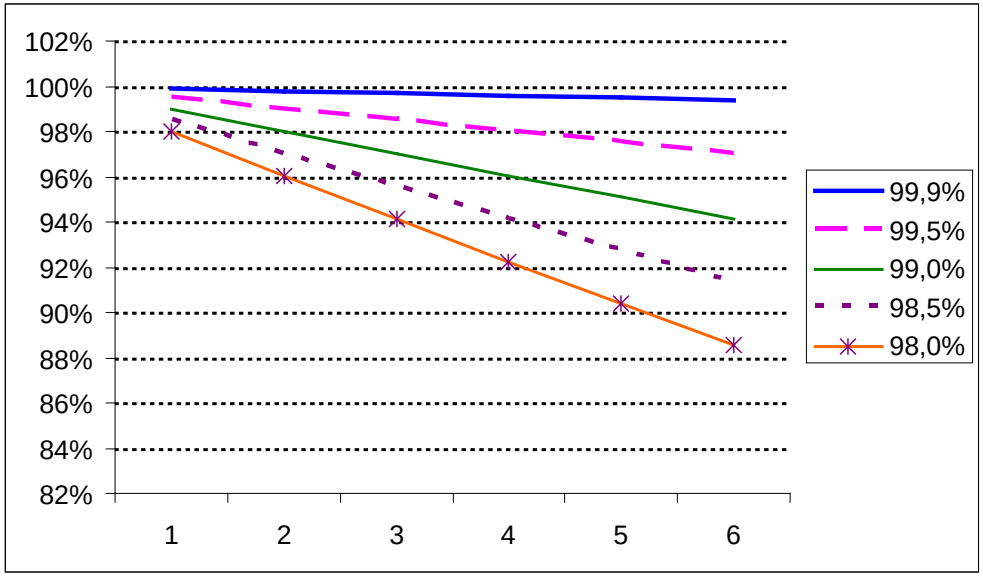
Verfügbarkeit berechnen



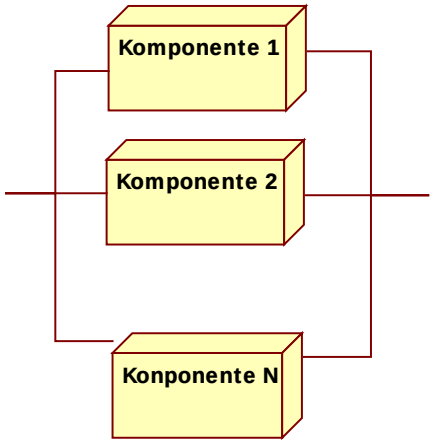
Mehrfachsystem ohne Redundanz

Verfügbarkeit (in %) = $A_1 * A_2 * A_3 \ldots * A_N$

Der Ausfall einer Komponente führt zum Ausfall des gesamten System



Verfügbarkeit berechnen



Einfachsystem mit mehrfacher Redundanz

- Jede Komponente erfüllt die gleiche Aufgabe

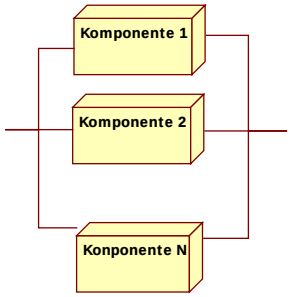
Das System als ganzes ist verfügbar solange einer der Komponenten verfügbar ist.

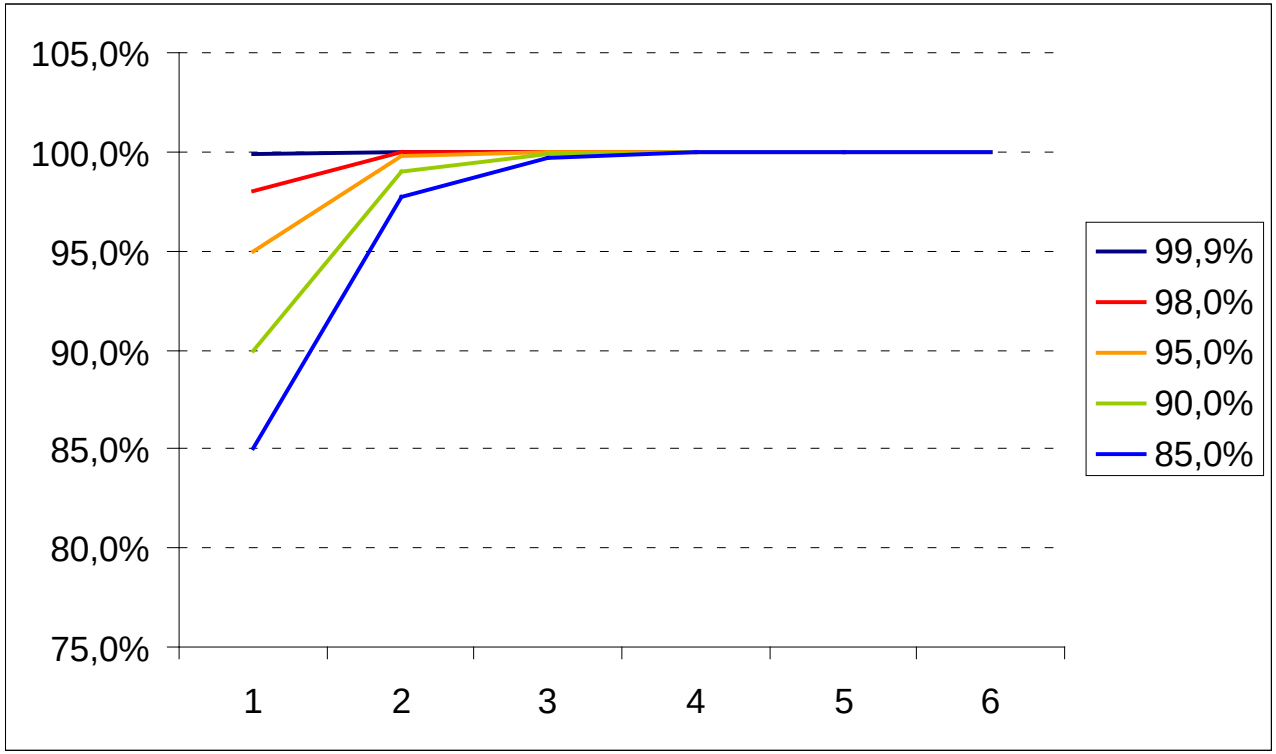
$$\text{Verfügbarkeit (in \%)} = 1 - \underbrace{(1 - A)^N}$$

Wahrscheinlichkeit dass alle Komponenten Gleichzeitig ausfallen

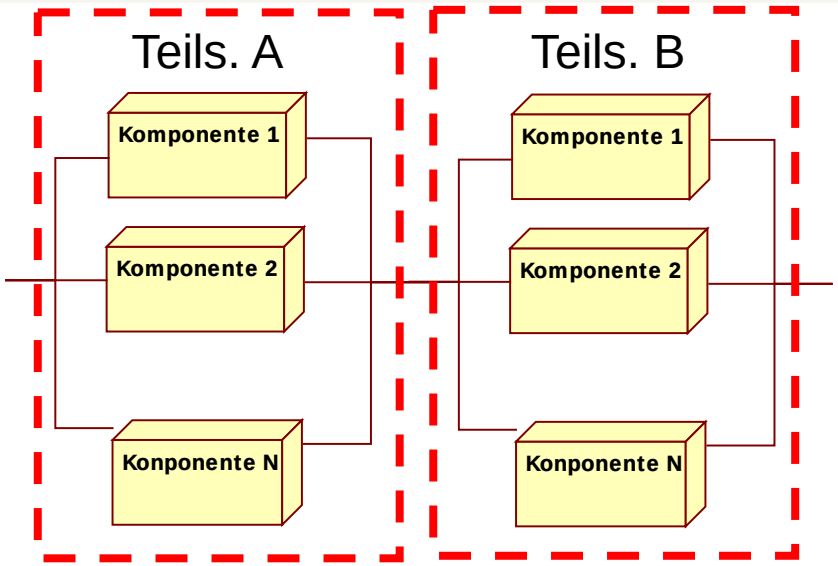
Hochverfügbarkeit und Downtime, Andrea Held

Verfügbarkeit berechnen





Verfügbarkeit berechnen – Kombination von mehreren Komponenten



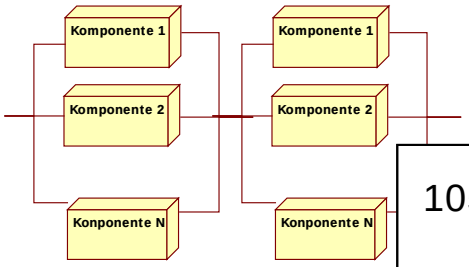
Mehrfachsystem mit
mehrfacher Redundanz

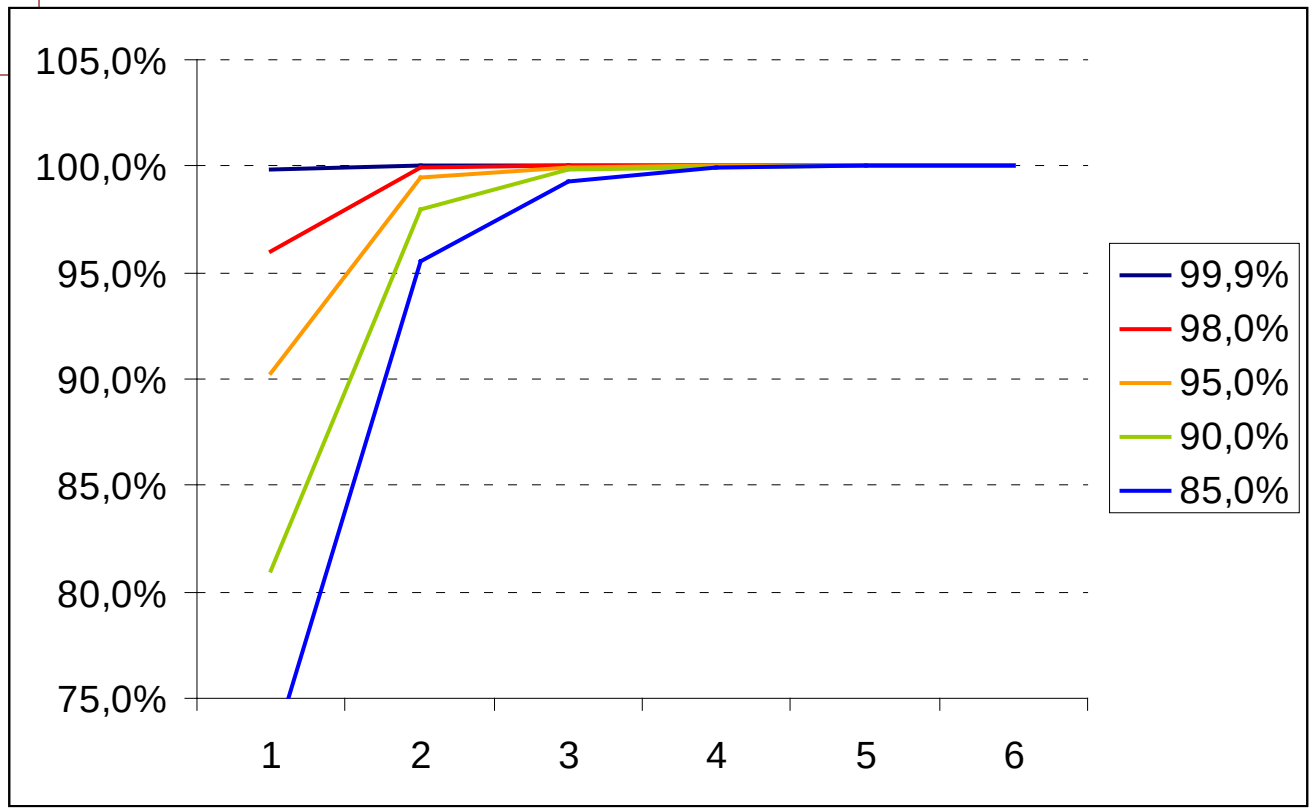
- Jede Komponente erfüllt die gleiche Aufgabe
- Verschiedene Systemteile sind mehrfach redundant ausgelegt

Das System als ganzes ist verfügbar solange einer der Komponenten verfügbar ist.

$$\text{Verfügbarkeit (in \%)} = \underbrace{(\,1 - (\,1 - A\,)^{N1}\,)}_{\text{Verfügbarkeit Teilsystem A}} * \underbrace{(\,1 - (\,1 - B\,)^{N2}\,)}_{\text{Verfügbarkeit Teilsystem B}}$$

Verfügbarkeit berechnen – Kombination von mehreren Komponenten





Verfügbarkeit berechnen – Kombination von mehreren Komponenten

- Ein komplexes System setzt sich in der Regel aus mehreren voneinander abhängige Komponenten zusammen.
- Eine hochverfügbare Auslegung nur von einzelnen Komponenten ist wenig effektiv:

  0,99 x 0,99 x 0,99 x 0,99 = 0,9606

  0,99 x **0,999** x 0,99 x 0,99 = 0,9693

  es sei denn, einzelne Komponenten haben eine signifikant niedrigere Einzelverfügbarkeit:

  0,99 x 0,99 x **0,95** x 0,99 = 0,9218

  0,99 x 0,99 x 0,99 x 0,99 = 0,9606

  Ziel muss eine Architektur ohne einen „Single Point of Failure“ sein, d.h. ALLE Komponenten sollten redundant (geclustert) ausgelegt sein

Prinzipien hochverfügbarer Anwendungen

Eigenschaften hochverfügbarer Anwendungen

- Redundante Auslegung aller Teilsysteme um einen Single Point of Failure zu eliminieren
- Toleranz und Transparenz gegenüber Fehlern
- Präventive Build-In-Funktionalitäten
- Proaktives Monitoring und schnelle Fehlererkennung
- Schnelle Wiederherstellungsmöglichkeiten
- Automatisierte Wiederherstellung ohne administrative Eingriffe
- Kein oder geringer Datenverlust

Prinzipien hochverfügbarer Anwendungen

Fehlertolerante Systeme

- Reaktion auf jede erdenkliche Art von Fehler
- Elimination eines Single Point of Failure in Hardware und Software
- Wichtig: Alle Elemente des Systems müssen Fehler-tolerant sein.

Prinzipien hochverfügbarer Anwendungen

Fehlertolerante Systeme: Hardware Fehlertoleranz

- Einsatz von hochverfügbaren Festplattensystemen z.B. RAID
- Unterbrechungsfreie Stromversorgung (USV) + Generatoren
- uvm.

Fehlertolerante Systeme: Software Fehlertoleranz

- Anwendung kann nach einem Fehler korrekt weiterarbeiten (Failover)
- Erkennen aktiver Transaktionen/Verarbeitungseinheiten
- Reaktion auf Fehlersituation und Bereinigung/Wiederholen offener Transaktionen

Prinzipien hochverfügbarer Anwendungen

Fehlertolerante Systeme: Hybride Systeme

- Kombination aus redundanter Hardware und Software
- Ganze Rechnersysteme werden redundante vorgehalten
- Intelligente Cluster-Software welche den Fehlerfall erkennt und einen Failover durchführt.
- Automatische Wiederherstellung des Systems
- Idealerweise ohne Verluste von Daten

Prinzipien hochverfügbarer Anwendungen

Maßnahmen bei geplante Downtime

- Hardware Upgrades: Verwendung von Hot Plug Hardware
- Software Upgrades: Rolling Upgrades über mehrere Clusterknoten hinweg
- Wartungsarbeiten: Switchover auf einen anderen Knoten im Cluster

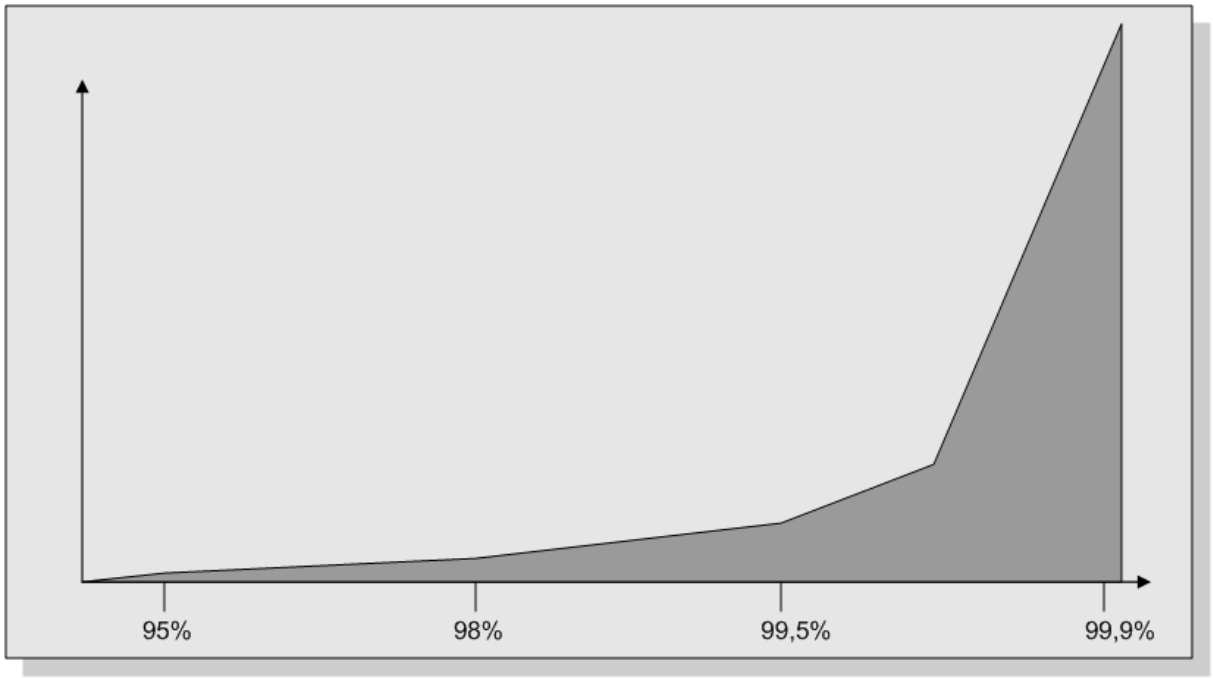
Prinzipien hochverfügbarer Anwendungen

Maßnahmen bei ungeplanter Downtime

- Defekte Hardware: Einsatz redundanter Hardware
- Fehlerhafte Software: Verwendung aktueller Patches
- Bedienungsfehler: Aktuelles und intaktes Backup, besser noch Standby System mit Delay in der Replikation
- Stromausfälle: getrennte Stromnetze, unterbrechungsfreie Stromversorgung
- Netzwerkprobleme: Redundante Auslegung der Netzwerkkomponenten
- Katastrophen / Desaster (Überflutung, Erdbeben, Bombenanschläge): Verwendung von Ausweichrechenzentren, Geo-Cluster, Geo-Spiegelung, Replikation

Prinzipien hochverfügbarer Anwendungen

Kosten der Hochverfügbarkeit



Kostensteigerungskurve nach Verfügbarkeit
(Quelle: Gartner Research)

Prinzipien hochverfügbarer Anwendungen

Redundante Auslegungen von Systemkomponenten – Auf verschiedenen Ebenen

1. *Hardware Clustering:* RAID is such technology used for both performance and reliable disks.

2. *OS (Server) Virtualization: We know many OS-level virtualization programs that can run many logical servers within same physical server at the same time.*

3. *DB Clustering: Many DBMS supports clustered database instances via different topologies. We can add JDBC-level cluster libraries to this category that simulate clustered database feature.*

https://dzone.com/articles/application-clustering

Prinzipien hochverfügbarer Anwendungen

Redundante Auslegungen von Systemkomponenten – Auf verschiedenen Ebenen

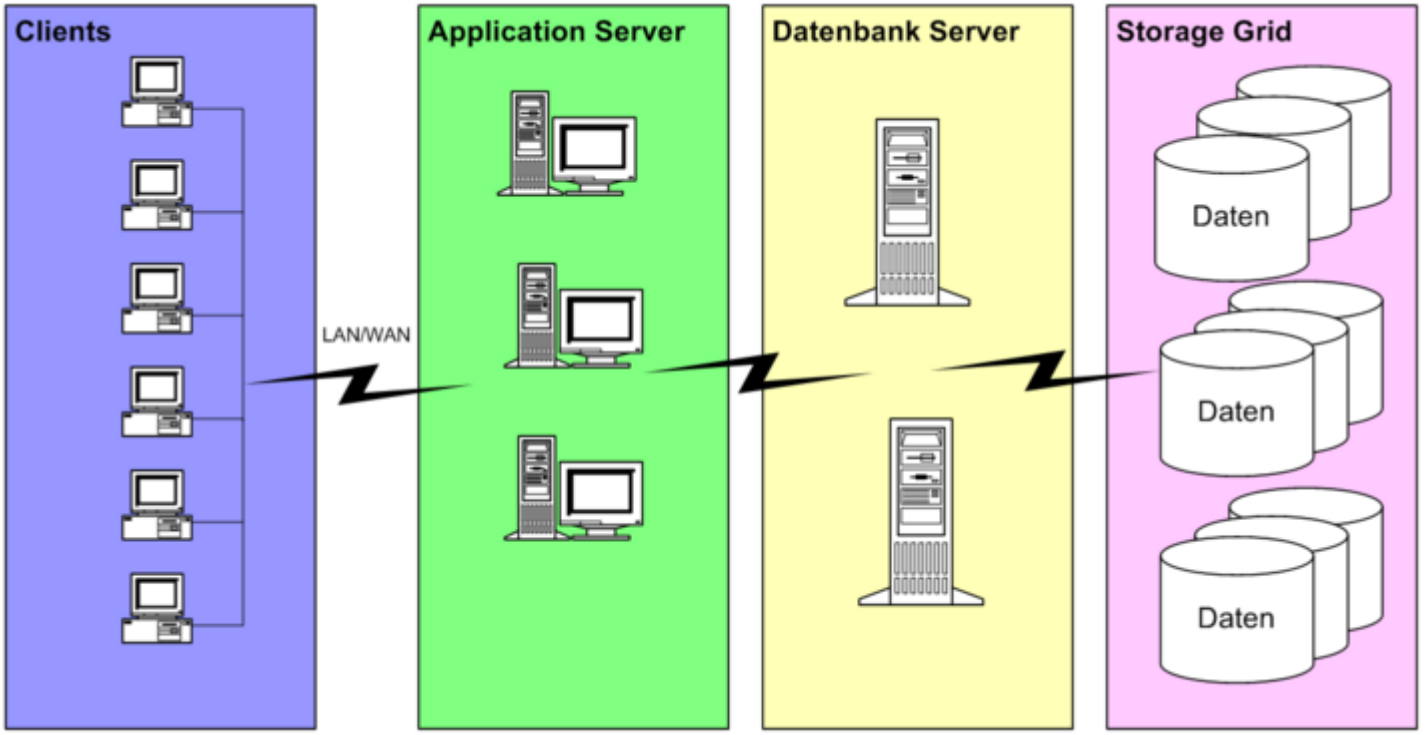
4. *AS Clustering: Application servers (and HTTP servers) support clustering with varying capabilities. Some of them can make session state replication across AS instances. Some of them have load-balancers to distribute coming requests. Some of them can have transparent fail-over feature etc.*

5. *Application Clustering:* Rework of the application itself so that it becomes cluster aware

https://dzone.com/articles/application-clustering

Prinzipien hochverfügbarer Anwendungen

Ein mögliches, hochverfügbares, hybrides Systeme



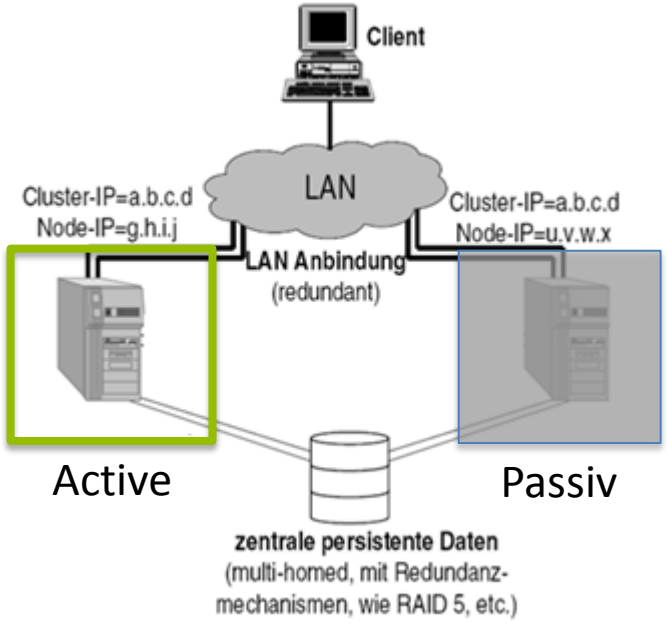
Verteilungsmuster „Standalone"

Standalone - Keine Redundanz

- Bei Ausfall eines Gerätes muss Ersatz erst beschafft, konfiguriert und in Betrieb genommen werden
- Mean Time To Revor (MTTR) = Tage bis Wochen
- Nicht akzeptabel für Informationssystem mit einer hohen Relevanz für die Wirtschaftliche Existenz einer Firma

Verteilungsmuster „Cluster"

Infrastruktur für einen einfachen Fail-Over-Cluster



- Der Cluster soll wie EIN System nach außen erscheinen (***Single System Image***)
- Dienste sollen über einen logischen Zugangspunkt genutzt werden können (z.B. IP-Adresse, Port- Nummer)
- im Fehlerfall eines Dienstes / einer HW-Komponente soll transparent und möglichst ohne Zeitverzögerung auf einen redundanten Dienst / eine redundante HW-Komponente umgeschaltet werden

Verteilungsmuster „Cluster"

Eigenschaften eines einfachen Fail-Over-Cluster

- Haupt- und Replikatserver sind vollständig identisch, liegen jedoch auf unterschiedlichen Knoten im verteilten System
- Im Normalfall arbeitet nur der Hauptserver
- Tritt ein Fehler auf, werden alle Aufrufe an den Replikatserver geleitet.
- Das System wirkt nach außen hin wie eine Einheit (z.B. erreichbar über die gleiche IP-Adresse)

Verteilungsmuster „Cluster"

Fail-Over-Cluster - Cold-Stand-By

- Ersatzgerät steht fertig konfiguriert bereit, befindet sich aber nicht in Betrieb und auch nicht in die Infrastruktur integriert.
- Es kann bei Ausfall gegen das primäre Gerät (manuell) ausgetaucht werden.
- Der Fehlerfall wird mit Hilfe eines geeigneten Systemmanagements erkannt.
- MTTR = mehrere Stunden

Verteilungsmuster „Cluster"

Fail-Over-Cluster - hot stand-by (active/passive oder primary/secondary)

- Ersatzgerät ist vollständig konfiguriert, in die Infrastruktur integriert und in Betrieb, wird jedoch im Leerlauf betrieben.
- **manual fail-over**
  - Im Fehlerfall des Primärgerätes übernimmt das Sekundärgerät durch manuelle Umkonfiguration des Betriebspersonals die Aufgaben des Primärgerätes.
  - Der Fehlerfall wird mit Hilfe eines geeigneten Systemmanagements erkannt.
  - MTTR < 1h

Verteilungsmuster „Cluster"

Fail-Over-Cluster - hot stand-by (active/passive oder primary/secondary)

- **automatic fail-over**
  - Fehlerfälle des Primärsystems werden durch eine Clustering-Software automatisch erkannt
  - Regelmäßige Prüfung ob der Knoten aktiv ist und lebt erfolgt über einen Heart-Beat: Möglichkeiten der Überwachung:
    - Liste von „verfügbaren“ Prozessen
    - Abfrage von Ports
    - Regelmäßige Pings
  - Diese Software übernimmt auch automatisch die Verlagerung der Arbeit auf das Sekundärsystem.
  - MTTR < 1 min

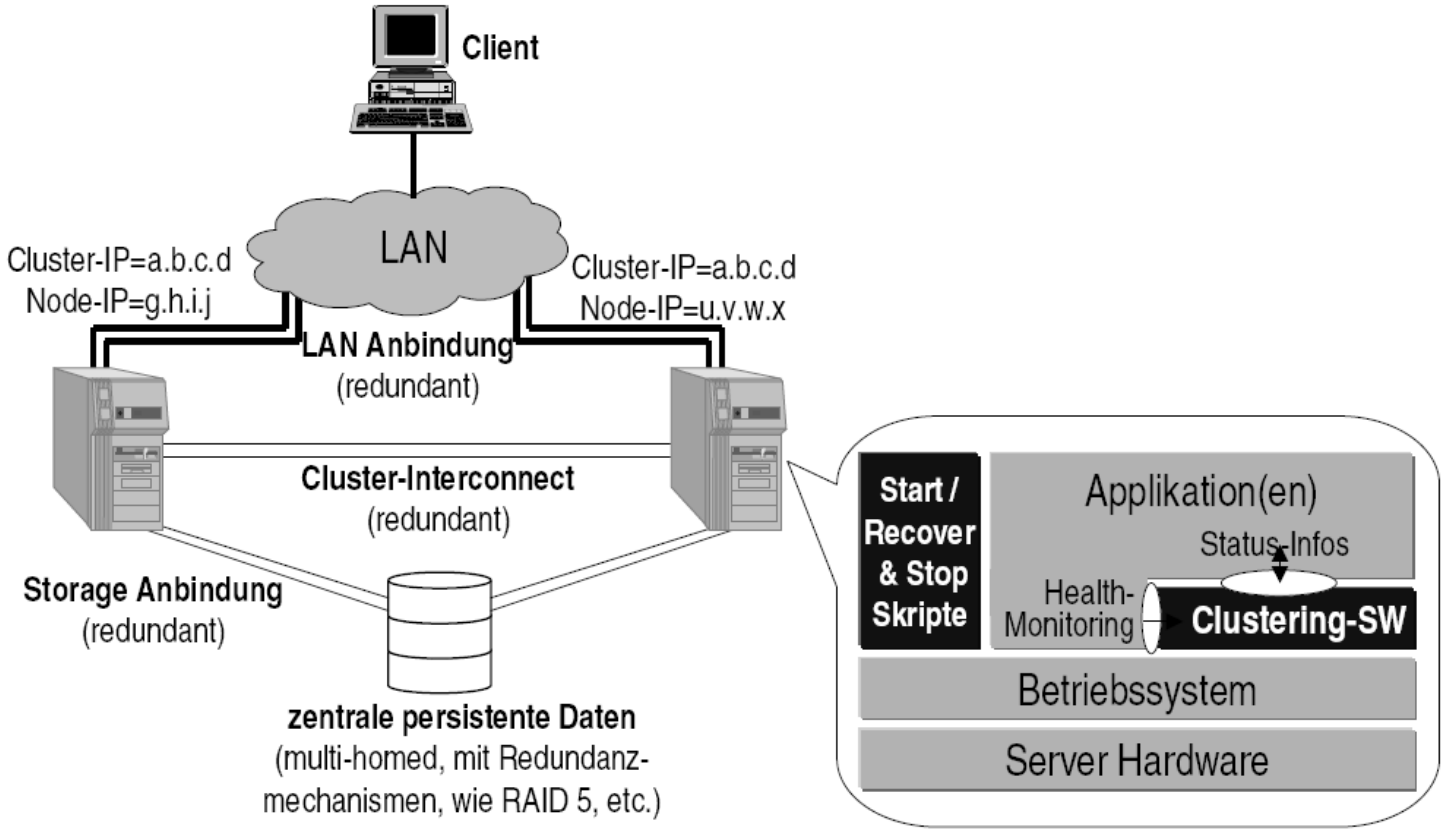
Verteilungsmuster „Cluster"

Voraussetzung für eine Applikation in einem Fail-Over-Cluster

- Die Nutzung der Applikation erfolgt über das LAN (nicht über Pipes o.ä .)
- Die Applikation lässt sich so konfigurieren, dass sie wesentliche Zustandsdaten auf dem zentralen persistenten Storage-System ablegt (und somit ein Recovery überhaupt möglich wird)
- Die Applikation ist verträglich mit der Systemkonfiguration, wie z.B.:
  - Server mit mehreren IP-Adressen / Netzwerkkarten
  - keine Abhängigkeit vom physikalischen Server-Namen

Verteilungsmuster „Cluster"

Infrastruktur für einen Fail-Over-Cluster mit hot-stand-by



Verteilungsmuster „Cluster"

Infrastruktur für einen Fail-Over-Cluster mit hot-stand-by

Aufgaben der Cluster-Software

- Erkennen das ein Konten ausgefallen oder nicht voll funktionsfähig ist
- Durchführen eines Fail-Overs
  - Stoppen des fehlerhaften Knotens
  - Start der des neuen Knotens
- Mannueller Fail-Over durch den Administrator

Verteilungsmuster „Cluster"

Infrastruktur für einen Fail-Over-Cluster – Step 2

Erkennen das ein Knoten/Anwendung ausgefallen ist

- Prüfen der Cluster-Software über das verteilte Cluster-Membership-Protokoll mit Hilfe von Heartbeat-Alive-Checks
- Prüfen der Betriebsumgebung: durch Monitoring von Systemresourcen wie z.B. CPU-Last, Swap-Space
- Prüfen der Anwendungen: durch Aufruf einer Monitoringschnittstelle oder z.B. Überwachen einer Prozessliste

Verteilungsmuster „Cluster"

Infrastruktur für einen Fail-Over-Cluster – Step 3

Fail-Over druchführen

- Anwendung auf den ursprünglichen Knoten beenden
  - Stop-Skripte
- Recover ausführen (optional)
- Anwendung auf dem neuen Knoten starten
  - Start-Skripte

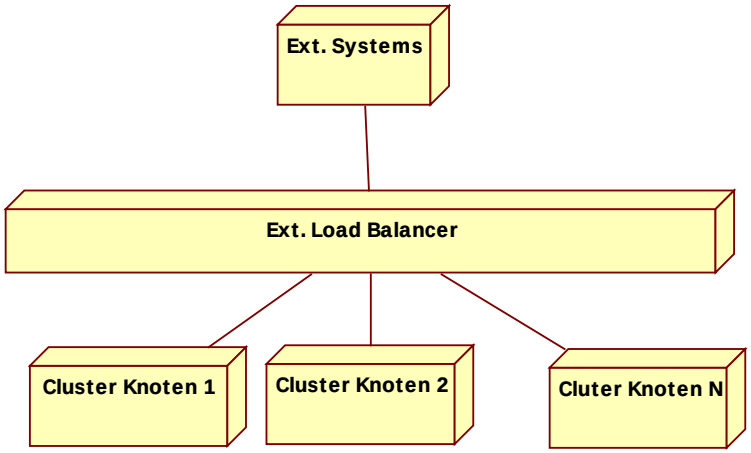
Verteilungsmuster „Cluster"

Load-Balancing-Cluster - active/active oder symmetrisches Cluster

- Mehrere identisch konfigurierte Geräte werden nebenläufig betrieben
- Die Geräte teilen sich die Arbeitslast untereinander auf
- Die Aufteilung erfolgt durch einen Loadbalancer (dynamisch oder statisch)

- Fehlerfälle in einer Komponten werden automatisch erkannt (z.B. durch Heardbeat-Alive-Checks).
  - Die Arbeitslast wird daraufhin nur noch zwischen den verbleibenden Geräten aufgeteilt.
  - MTTR < 1 min

Verteilungsmuster „Cluster“ (Load-Balancing)



Lastverteilung durch Load-Balancer (Request-Ebene)

- Der Load-Balancer verteilt eingehende Anfragen auf verschiedene Cluster-Knoten
- Die Cluster Knoten arbeiten komplette Request autark (unabhängig voneinander) ab
- Achtung: Auch der Load Balancer sollte redundant ausgelegt werden. Ansonsten existiert ein „Single-Point-Of-Failure“
- Typische Architektur für Web-Anwendungen.

Applikations-Server-Cluster und deren Besonderheiten

Redundante Application-Server

- Folgende Ziele wurden bis jetzt erreicht:
  - Sicherstellung der Verfügbarkeit von Servern/Systemen
  - Lastverteilung auf die unterschiedlichen Komponente auch bei Ausfall einer Komponente
  - Aber: Bei Ausfall einer Komponenten werden die auf dieser bearbeiteten Sessions abgebrochen (evtl. in einem inkonsistenten Zustand).
  - Erst für neue Sessions greift die Cluster-Redundanz in Zusammenhang mit der Lastbalancierung

  - Wie kann ein Session-Fail-Over realisiert werden ??

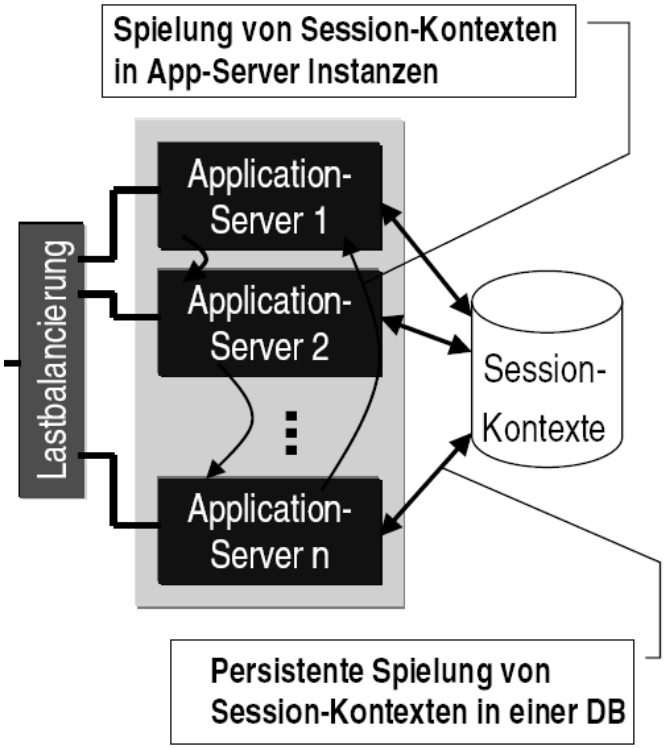
Applikations-Server-Cluster und deren Besonderheiten

Redundante Application-Server – Session Fail Over

- Session Fail Over:
  - Sicherstellung der Fortsetzbarkeit einer Session selbst wenn der aktuell bearbeitende Server / Dienst ausfällt
- Notwendige Eigenschaften
  - persistente Speicherung der Session-Kontexte außerhalb des eigentlich bearbeitenden Dienste-Server
  - automatischer Zugriff auf diese Kopie eines Session-Kontextes, wenn ein Request/Response einer noch unbekannte Session an einem Dienste-Server ankommt

Applikations-Server-Cluster und deren Besonderheiten

Redundante Application-Server – Session Fail Over



- Variante A: Spiegelung des Session-Context über eine Datenbank:
- Variante B: Spiegelung des Session-Context über verschieden Konten (im Haupsteicher):

Applikations-Server-Cluster und deren Besonderheiten

Redundante Application-Server – Session Fail Over

- Spiegelung des Session-Context über verschieden Konten (im Haupspeicher):
  - aus Performanzgründen in nur je einem weiteren Server-Knoten pro
  - Primary und Secondary Server sind aus Session-ID ersichtlich
  - bei einem Fail-Over wird nächster Request durch den Load-Balancer auf einen beliebigen neuen Application-Server geroutet, der zum neuen Primary wird und den Session-Kontext vom Secondary lädt;

Applikations-Server-Cluster und deren Besonderheiten

Redundante Application-Server – Session Fail Over

Bewertung:

- Spiegelung der Session-Kontexte in einer Datenbank beeinflusst die Antwortzeit negativ
- Spiegelung der Session-Kontext auf anderen Knoten löst das Problem nicht endgültig (Extermfall: Ausfall beider Knoten)
- Evtl. Beide Ansätze kombinieren
- Kritisch prüfen ob der Session Fail Over notwendig ist
  - Evtl. ausreichend, wenn einzelne Session-Aktionen unter Transaktionssemantik abgewickelt werden, um keinen inkonsistenten Zustände bei Session-Abbruch zu hinterlassen

Applikations-Cluster und deren Besonderheiten

Software-Komponenten müssen auf ihre Clustertauglichkeit hin untersucht werden.

Grundsätzliche Problem(e):

Zustand/State: Bindung von Daten an die Ressourcen des verarbeitend Cluster-Knotens

- Die Komponenten einer Applikation müssen so überarbeitet werden so dass die Integrität der Zustandsinformation gewährleistet ist.

https://dzone.com/articles/application-clustering

Applikations-Cluster und deren Besonderheiten

Typische Problembereiche

1. Locking: Application locks should be shared across different cluster nodes. If DB locks are used then it is cluster-safe. If a singleton lock service is doing locking, then we need to make lock acquire-release calls among cluster nodes synchronously. For example; a user is running a process on AS cluster node1 and hold a pessimistic lock on Entity A, then another user's process running on cluster node2 must receive an exception when trying to update Entity A.

2. Caches: Every cache service should be analyzed if a cluster-enablement is required. Some caches may repeat on cluster nodes but that may not cause any problem. On the other hand, let's say we have a persistent object cache service that runs according to update information to tables and checks if any table modified. A process on another cluster may update a table and that change information should be reflected to all caches on the other cluster nodes.

https://dzone.com/articles/application-clustering

Applikations-Cluster und deren Besonderheiten

Typische Problembereiche

3. Events: You may have designed some type of application events and those may need to be cluster-aware. For example, you have a notification event that triggers when a user sends a message to another user. If that event occurs in node01 and other user is on node02, he would never receive that event and message if that event is not cluster-enabled.

4. Identities: Let's say we have a sequence generator service that has a cache to prevent accessing database every time a number is requested. If identity service is not cluster-enabled, identity services on different cluster nodes would generate same identity numbers and applications would get duplicate record errors.

https://dzone.com/articles/application-clustering

Applikations-Cluster und deren Besonderheiten

Typische Problembereiche

5. Authentication: If user can login on each cluster node then some authentication information need to be shared. One example is a feature such that a user account is locked after multiple failed login tries. If failed attempts are not shared, a hacker may continue his attempts on other nodes. I don't add authorization here since we didn't need an adjustment for it to work in clustered environment (Your authorization service may need it).

6. Scheduling: Scheduled or batched tasks should be considered when clustering. Same scheduled processes may try to run on each instance whereas only one execution is enough.

https://dzone.com/articles/application-clustering

Applikations-Cluster und deren Besonderheiten

Typische Problembereiche

7. Logging (File IO): You have to deal with file operations in a clustered environment. One file issue is log files. Multiple cluster nodes may try to write the same log file and make it corrupt or cause IO errors if not cluster-enabled. Our solution for this is to generate and use different log files for each cluster node.

8. Messaging (Network IO): If you are using some messaging services, you have to make sure that message service is cluster-enabled. If you are using a JMS product, you need to test in a clustered environment whether it behaves appropriately. If you wrote it and you plan to use multiple messaging service instances, then you have to use different ports if cluster nodes are on the same server. We just used only one instance of messaging service to overcome that kind of problems.

https://dzone.com/articles/application-clustering

Applikations-Cluster und deren Besonderheiten

Lösungsansätze

State Replication: You may have multiple service instances on cluster nodes and synchronize the state by replicating it. There are many replication techniques you can use; port communication, RMI, JavaSpaces. This architecture has advantage in fail-over and disadvantage in performance.

Hub-and-Spoke (Master-Slave): In this architecture, you would have only one instance of services on master cluster node and child nodes will invoke master node services through for instance RMI method calls. This architecture has advantage in performance but disadvantage in fail-over capability (if master server fails, child nodes would lose service states).

Remove Conflicts: Make the involved components cluster capable by removing all the conflicts. This also increases the throughput

https://dzone.com/articles/application-clustering

Fazit – Physische Verteilung

- Ausgehend von den (nicht-funktionalen) Anforderungen ist zu entschieden wie die Applikation zu verteilen ist.
- Je höher die nicht-funktionalen Anforderungen sind desto aufwendiger wird die physische Struktur
- Oftmals ist es ausreichend die Datenbank zu clustern. Der Anwendungsserver wird über eine Virtualisierungs-Lösung betrieben
- Achtung: Die Umsetzung nicht-funktionaler Anforderung kann sehr teuer sein.