Verteilte Systeme, 3. Teil

L. Lastverteilung

Einführung

- **Last in einem Rechensystem**

  tritt auf als
  - ▻ **Rechenlast**
  - ▻ **Speicherlast**

- **Lastverbund**

  - ▻ Zusammenfassung mehrerer Rechner(systeme) zur Aufnahme der **aggregierten Last**

  - ▻ **Verteilung** der Gesamtlast in geeigneter Weise auf die verschiedenen Rechner.
    Die die Last aufnehmenden Rechner werden auch als **Server** bezeichnet :
    - ▸ Verteilung der Rechenlast auf **Rechenserver**
    - ▸ Verteilung der Speicherlast auf **Dateiserver**

  - ▻ **Speicherlastverbunde** werden durch **verteilte Dateisysteme** realisiert (s. Kap. "Verteilte Dateisysteme")

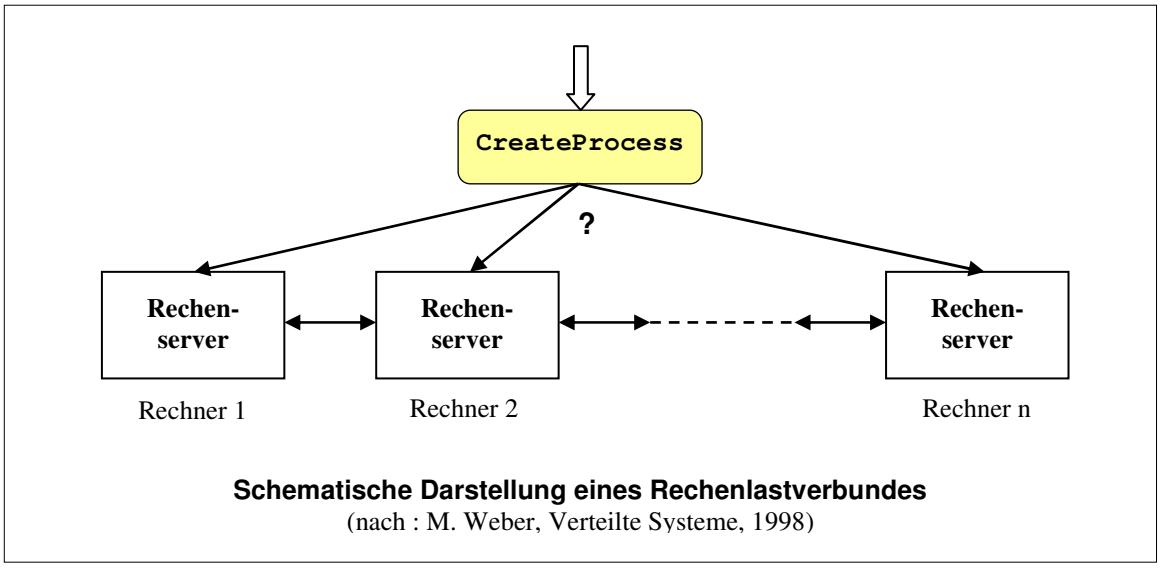
  - ▻ Hier liegt der Schwerpunkt auf der Betrachtung von **Rechenlastverbunden**

- **Fragestellungen**

  - ▻ Soll ein neu zu erzeugender Prozess lokal oder entfernt ausgeführt werden? (Übertragungs-Strategie)

  - ▻ Wo soll ein neu zu erzeugender Prozess tatsächlich kreiert werden? Wenn entfernt, wo?
    (Positions-Strategie)

  - ▻ Soll ein bereits laufender Prozess gegebenenfalls zu einem anderen Rechner wechseln? Wenn ja, zu welchem?
    (Migrations-Strategie)



Schematische Darstellung eines Rechenlastverbundes
(nach : M. Weber, Verteilte Systeme, 1998)

- **Systemmodelle für Lastverbunde**

  - ▻ Die Rechner bzw CPUs eines Lastverbundes lassen sich auf unterschiedliche Weise zusammenfassen und anordnen

  - ▻ Zwei wichtige Systemmodelle für Lastverbunde sind:
    - ▸ **Workstation-Modell**
    - ▸ **Prozessorpool-Modell**

Lastverbund – Workstation-Modell (1)

- **Eigenschaften**

  ◇ Ein Lastverbund nach dem Workstation-Modell basiert auf **vernetzten** privaten oder öffentlichen **Workstations**.

  ◇ An jeder Workstation ist zu einem bestimmten Zeitpunkt **höchstens ein Benutzer** angemeldet.
  → (temporärer) "Besitzer" der Workstation

  Jede Workstation befindet sich in einem von zwei möglichen **Zuständen**:

  - ▻ **belegt** : Ein Benutzer ist angemeldet
  - ▻ **frei** (leerlaufend, *idle*) : Kein Benutzer ist angemeldet

  ◇ **Vorteile**:

  - ▻ einfaches, leicht zu verstehendes Modell.
  - ▻ Benutzer verfügen über feste Ressourcen.
  - ▻ Auf Benutzereingaben wird mit festen Antwortzeiten reagiert
  - ▻ Benutzer haben hohe Autonomie.
    Sie können die Ressourcen der Workstation nach ihren jeweiligen Bedürfnissen einsetzen.

  ◇ **Nachteile**:

  - ▻ Ressourcen sind beschränkt, auch wenn mehr gebraucht würden.
  - ▻ Ressourcen bleiben ungenutzt, wenn der Benutzer gerade nichts oder wenig tut
    oder die Workstation im Zustand *idle* ist

  ◇ Beobachtungen (an Universitäten) haben ergeben, dass selbst zu normalen Arbeitszeiten bis zu 30% der Workstations frei sind, d. h. nicht benutzt werden.

  ◇ **Idee:**

  - ▻ die **ungenutzte Rechenleistung** durch geeignete Mechanismen denen, die sie benötigen, **verfügbar** machen.
  - ▻ überlastete Workstations entlasten und „freie“ Workstations belasten. → **bessere Lastverteilung**
    → die "**freien**" Workstations werden zu **Servern**, die **überlasteten** zu ihren **Clients**.
  - ▻ Beispielsystem: Condor.
    (M. J. Litzkow, M. Livny, M. W. Mutkas : Condor – A Hunter of Idle Workstations;
    8th IEEE International Conference on Distributed Computing Systems, 1988)

  ◇ **Schlüsselaspekte zur Umsetzung obiger Idee** :

  - ▻ Wie können leerlaufende Workstations gefunden werden?
  - ▻ Wie kann ein entfernt ablaufender Prozess transparent ausgeführt werden?
  - ▻ Was passiert, wenn sich ein Benutzer lokal an der Workstation anmeldet?

Lastverbund – Workstation-Modell (2)

• Kriterien zur Erkennung "arbeitsloser" Workstations

◇ Die Überprüfung, ob ein Benutzer an der Maschine angemeldet ist, genügt i.a. nicht :

- ▻ Selbst wenn kein Benutzer angemeldet ist, können viele Hintergrundprozesse (Dämonen) laufen
- ▻ Ein angemeldeter Benutzer muss nicht unbedingt die Maschine belasten.

◇ Häufig wird eine Workstation als "arbeitslos" betrachtet, wenn

- ▻ **keine benutzer-initiierten Prozesse** laufen oder
- ▻ die Benutzerprozesse keine wesentliche Prozessorlast erzeugen und über einen gewissen Zeitraum (z.B. mehrere Minuten) **keine Benutzerinteraktionen** (Tastatur- oder Mauseingaben) stattfinden

• Auffinden leerlaufender Workstations

◇ **Client-gesteuert**

- ▻ Broadcast-Anfrage an alle anderen Workstations im Netz (potentiellen Rechenserver) unter Angabe des entfernt auszuführenden Programms einschließlich der hierfür benötigten Ressourcenanforderungen.
- ▻ Angabe der Ressourcenanforderungen ist nur notwendig, wenn nicht alle Workstations gleich aufgebaut sind und nicht jedes Programm auf allen Workstations ausgeführt werden kann.
- ▻ Die "arbeitslosen" Workstations, die das Programm ausführen können, melden sich zurück
- ▻ Der anfordernde Client wählt eine Workstation aus, der er den Auftrag zur Ausführung des Programms erteilt
- ▻ In einer sehr praktischen Variante verzögern die "arbeitslosen" Workstations ihre Antwort etwas, wobei die Verzögerungsdauer proportional zu ihrer aktuellen Auslastung ist.
  Dadurch kommt die Antwort von der am wenigsten ausgelasteten Workstation als erste zurück.
  Diese kann dann ausgewählt werden.

◇ **Server-gesteuert**

▷ **zentrale Variante**

- ▻ Eine in den Zustand *idle* übergehende Workstation teilt dies unter Angabe ihrer Identifikation, ihrer Eigenschaften und ihrer Ressourcen einer **zentralen Registratur** (Eintrag in Registrierungsdatei oder Datenbank). mit.
- ▻ Ein Client, der nach einer "arbeitslosen" Workstation sucht, wendet sich an die Registratur und bekommt von dieser gegebenenfalls die Identifikation einer geeigneten Workstation mitgeteilt.

▷ **dezentrale Variante**

- ▻ Eine in den Zustand *idle* übergehende Workstation teilt dies per Broadcast allen anderen Workstations im Netzwerk mit.
- ▻ Die Workstations tragen die freie Workstation in eine eigene lokal geführte Registrierungsdatei ein.
- ▻ Bei Bedarf wählt ein Client seinen Server aus seiner lokalen Registrierungsdatei aus.

▷ **Problem** (bei beiden Varianten)

- ▻ Gleichzeitige Auswahl einer Workstation als potentiellen Server durch zwei (oder mehr) Clients
- ▻ Abhilfe:
  - Client meldet sich bei der ausgewählten Workstation an.
  - Diese meldet sich, falls sie noch zur Verfügung steht, aus der – zentral oder dezentral geführten – Registratur ab
  - und sendet dem anfordernden Client eine Belegungsbestätigung
  - Der Client erteilt der Workstation den Bearbeitungsauftrag.

Lastverbund – Workstation-Modell (3)

- **Transparente Ausführung eines entfernt erteilten Rechenauftrags**

◇ **Ortstransparenz für den Benutzer**
Für den Benutzer, dessen Workstation Rechenlast auslagert, soll es transparent sein, wo die Last bearbeitet wird.
→ es darf für ihn keinen Unterschied zwischen der lokalen und der entfernten Ausführung eines Prozesses geben.

◇ **Umgebungstransparenz für den auszuführenden Prozess**
Der entfernte Prozess muss auf seinem Wirts-Rechner die **gleiche Umgebung** wie auf seiner Heimat-Workstation vorfinden.
→ Er muss somit dieselben Berechnungen ausführen, die er auch lokal ausgeführt hätte.
Hierfür benötigt der Prozess:

- ▻ die gleiche Sicht auf das Dateisystem,
- ▻ das gleiche Arbeitsverzeichnis,
- ▻ dieselben Umgebungsvariablen,
- ▻ gleichwertige Ressourcen.

◇ **Ausführung von Bibliotheksfunktionen und Systemaufrufen**

- ▻ Bei Bibliotheksfunktionen oder Systemaufrufen muss entschieden werden, ob sie vom Wirts- oder vom Heimat rechner bearbeitet werden:
- ▻ **Lokale Ausführung**
  - Alle Ein-/Ausgabeoperationen des Benutzers müssen auf der Heimat-Workstation ausgeführt werden, dies betrifft alle Aktionen bzgl. Tastatur, Maus, Bildschirm und Lautsprecher.
  - Das gleiche gilt für den Zugriff zu lokal auf der Heimat-Workstation gespeicherten Dateien.
- ▻ **Entfernte Ausführung**
  - Alle Systemaufrufe, die sich auf CPU-Zugriffe oder auf Hauptspeicherzugriffe zurückführen lassen, müssen auf dem Wirts-Rechner ausgeführt werden.
  - Temporäre Dateien sollten ebenfalls auf dem Wirts-Rechner angelegt werden.
    → größere Effizienz wegen geringerer Netzwerkkommunikation

- ▻ **Problem** :
  - Funktionen, die sich auf die **Zeit** beziehen. erfordern i.a. eine **systemweite globale Zeit** oder **zusätzliche Synchronisationsmaßnahmen**.
  - Sind die Uhren auf dem Heimat- und dem Wirts-Rechner nicht synchronisiert, kann die Ausführung von zeitabhängigen Programmen auf dem Wirts-Rechner fehlerhafte Ergebnisse erzeugen.
  - Eine Weitergabe aller zeitabhängigen Aufrufe an den Heimat-Rechner verursacht dagegen durch die notwendige Übertragung Verzögerungen, die zu weiteren Problemen führen können.

Lastverbund – Workstation-Modell (4)

- **Lokale Beendigung der "Arbeitslosigkeit" des Wirts-Rechners**

◇ Der **ursprüngliche lokale Leerlauf** der entfernt genutzten Workstation wird **aufgehoben**

- ▻ durch das **Anmelden** eines **lokalen Benutzers** ("Besitzers") der Workstation
- ▻ durch ein **Ereignis**, das **wartende Prozesse anstößt**

◇ **Behandlung des auf die Workstation ausgelagerten Prozesses** :

▷ **Prozess läuft weiter**

- ▻ Für den Gastprozess äußert sich außer einer Reduzierung der Arbeitsgeschwindigkeit nichts.
- ▻ Der lokale Benutzer wird durch die vom Gastprozess verursachte Lasterhöhung ebenfalls beeinträchtigt
  → Verlust der garantierten Antwortzeiten, Widerspruch zur Idee der "persönlichen" Workstation
  → Lokaler Benutzer stellt gegebenenfalls seine Workstation nicht mehr für den Lastverbund zur Verfügung.

▷ **Prozess wird abgebrochen**

- ▻ abrupter Abbruch ohne Warnung
- ▻ die bis zu diesem Zeitpunkt verrichtete Arbeit geht verloren, wenn nicht Zwischenergebnisse gesichert worden sind

▷ **Prozess terminiert nach Warnung**

- ▻ Der Prozess wird durch ein Signal über den bevorstehenden Abbruch informiert
- ▻ Der Prozess kann Zwischenergebnisse und –zustände sichern (Schreiben von Datenpuffern in Dateien, Schließen der Dateien usw.) und sich dann selbst gezielt beenden.

▷ **Prozess migriert**

- ▻ Der Prozess wird auf eine andere Workstation verlagert (entweder zurück zur Heimat-Workstation oder auf eine anderen "arbeitslosen" Rechner)
- ▻ Der Prozess muss dabei seine gesamte Umgebung "mitnehmen".
  U.a. müssen alle offenen Dateien und Kommunikationskanäle, laufende Timer, Warteschlangen mit angekommenen Nachrichten und alle prozessspezifischen Kernel-Datenstrukturen vom bisherigen Wirts-Rechner entfernt und auf dem Ziel-Rechner im alten Zustand wieder hergestellt werden.
  → komplizierte und schwierige Aufgabe

◇ **Zustand des Wirtsrechners nach Beendigung eines Gastprozesses**

Unabhängig von der Art seiner Beendigung muss ein Gastprozess seinen **Wirtsrechner so hinterlassen**, wie er ihn **vorgefunden** hat :

- ▻ Neben dem Gastprozess müssen auch alle seine Kindprozesse, deren Kindprozesse usw. den Wirtsrechner verlassen
- ▻ Alle temporären Dateien müssen entfernt werden.
- ▻ Alle Mailboxen und sonstigen vom Prozess verwendeten Datenstrukturen müssen gelöscht werden.
- ▻ Alle Kommunikationskanäle sind so einzustellen, dass spätere für den Prozess ankommende Nachrichten ignoriert oder umgeleitet werden.

Lastverbund – Prozessorpool-Modell (1)

- **Eigenschaften**

  ◇ Völlig anderer Ansatz zur Realisierung eines Lastverbundes .

  ◇ Die Gesamtrechenleistung wird **zentral** durch einem **Pool von Prozessoren** zur Verfügung gestellt.

  ◇ Die Benutzer (Verbraucher der Rechenleistung) interagieren mit dem Pool über leistungsfähige Graphikterminals

  ◇ Den Benutzern werden auf Anforderung je **nach Bedarf** eine passende Anzahl von **Prozessoren zur Verfügung gestellt**
  → **dynamische** Zuteilung

  ◇ Welche und wieviel Prozessoren jeweils zur Verfügung gestellt werden, ist **zum Benutzer hin transparent** und kann von diesem i.a. nicht beeinflusst werden.

  ◇ Analog zu einem Dateiserver mit mehreren Festplatten kann ein Prozessorpool als ein **Rechenserver** mit **mehreren Prozessoren** aufgefasst werden.

  ◇ Im Prinzip stellt dieser Rechenserver eine **Menge "leerlaufender" Workstations** zur Verfügung, auf die **dynamisch** zugegriffen werden kann.
  Jedem Benutzer kann temporär so viel Rechenleistung zugewiesen werden, wie er aktuell benötigt.
  Wenn er sie nicht mehr benötigt, gibt er sie wieder an den Pool zurück, so dass sie von anderen Benutzern angefordert werden kann.
  Es gibt jedoch keinen "Besitzer" einer derartigen "Workstation".

- **Eignung**

  ◇ Grundsätzlich gilt, dass eine **zentral zur Verfügung gestellte Rechenkapazität** im Mittel **effektiver genutzt** werden kann als die gleiche verteilt bereitgestellte Kapazität.
  Mit Hilfe der Warteschlangen-Theorie lässt sich nachweisen, dass beim **Ersetzen** von **`n`** **Einzelsystemen** mit jeweils gleicher Leistungsfähigkeit (z.B. Workstations) durch ein großes **System mit der `n`−fachen Leistungsfähigkeit** eines Einzelsystems (Prozessorpool), die **mittlere Antwortzeit** auf Bearbeitungsanfragen auf den **`n`−ten Teil sinkt**.

  ◇ Diese Effektivitätssteigerung gilt aber nur, wenn die zu bearbeitenden Aufgaben **ausreichend parallelisierbar** sind, so dass tatsächlich immer alle Prozessoren arbeiten können.

  ◇ Das Prozessorpool-Modell ist besonders geeignet für die Bearbeitung **rechenintensiver Aufgaben**, die einen **hohen Parallelisierungsgrad** besitzen.

  ◇ Die – eigentlich gegen verteilte Systeme sprechende – mittlere Antwortzeit ist nicht das einzige Beurteilungskriterium zur Eignung eines Rechnersystems.
  Andere wichtige Faktoren lassen wiederum verteilte Systeme effizienter erscheinen, z.B. :
  - Kosten-Leistungs-Verhältnis
  - Zuverlässigkeit und Fehlertoleranz
  - Flexibilität
  - geringe Schwankungen der Antwortzeit statt möglichst kurzer mittlerer Antwortzeit

- **Kombination von Workstation- und Prozessorpool-Modell**

  Wenn man graphischen Terminals durch Workstations mit graphischen Fähigkeiten ersetzt, lassen sich **beide Last-verteilungs-Modelle** auch **miteinander kombinieren** :

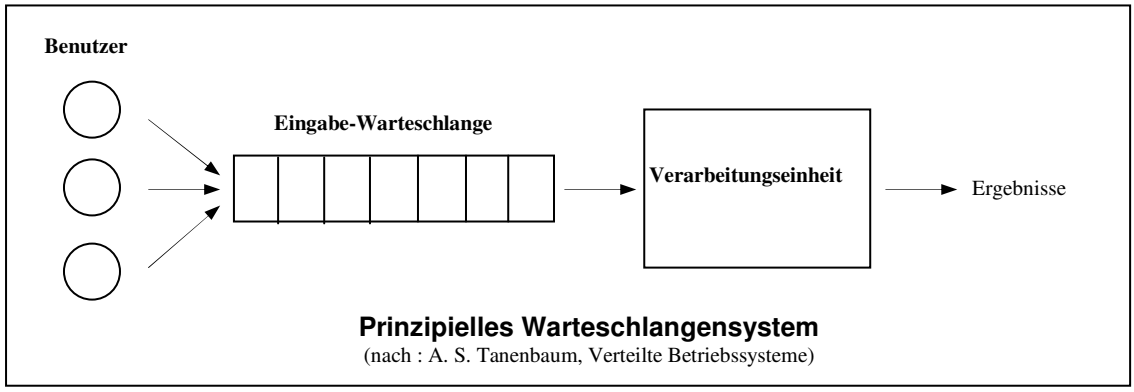
  ▻ Rechenintensive Anwendungen werden auf dem Prozessorpool abgearbeitet

  ▻ Anwendungen mit viel Benutzerinteraktion (z.B. Editoren) werden auf den Workstations ausgeführt.

Lastverbund – Prozessorpool-Modell (2)

- **Mittlere Antwortzeit eines Rechnersystems**

◇ Ein Rechnersystem, an das in beliebiger Folge Bearbeitungsanfragen gestellt werden, kann als **Warteschlangensystem** aufgefasst werden.
Die mittlere Antwortzeit eines derartigen Systems auf eine Bearbeitungsanfrage lässt sich mit Hilfe der **Warteschlangentheorie** ermitteln.



Prinzipielles Warteschlangensystem
(nach : A. S. Tanenbaum, Verteilte Betriebssysteme)

◇ Vorausgesetzt, ein Rechner kann **Anfragen** in einer ausreichend großen **Wartesschlange zwischenspeichern**, dann gilt für die **mittlere Antwortzeit T zur Beantwortung einer Anfrage** :

λ mittlere Eingaberate (Anfragen/sek)
μ mittlere Verarbeitungsrate (bearbeitete Anfragen/sek)
T mittlere Antwortzeit

$$T = \frac{1}{\mu - \lambda}$$

Beispiel :
Ein Server, der 50 Anfragen pro Sek. bearbeiten kann (μ), erhält im Mittel 40 Anfragen pro sek (λ).
Seine mittlere Antwortzeit beträgt 1/10 sek = 100 msek.

◇ Wenn n gleiche Workstations mit obigen Parametern existieren, gilt diese Beziehung für jede Workstation

◇ Werden die CPUs der n Workstations zu einem **Prozessorpool** zusammengefasst, so steigen sowohl die mittlere Verarbeitungsrate als auch – bei gleichem Benutzerverhalten – die mittlere Eingaberate auf den n-fachen Wert.
Damit ergibt sich für das kombinierte System die **mittlere Antwortzeit Tp** zu :

$n\lambda$ mittlere Eingaberate (Anfragen/sek)
$n\mu$ mittlere Verarbeitungsrate (bearbeitete Anfragen/sek)
Tp mittlere Antwortzeit

$$Tp = \frac{1}{n\mu - n\lambda} = T/n$$

$\Rightarrow$ die mittlere Antwortzeit ist auf den n-ten Teil gesunken.

Lastverteilungsverfahren - Überblick

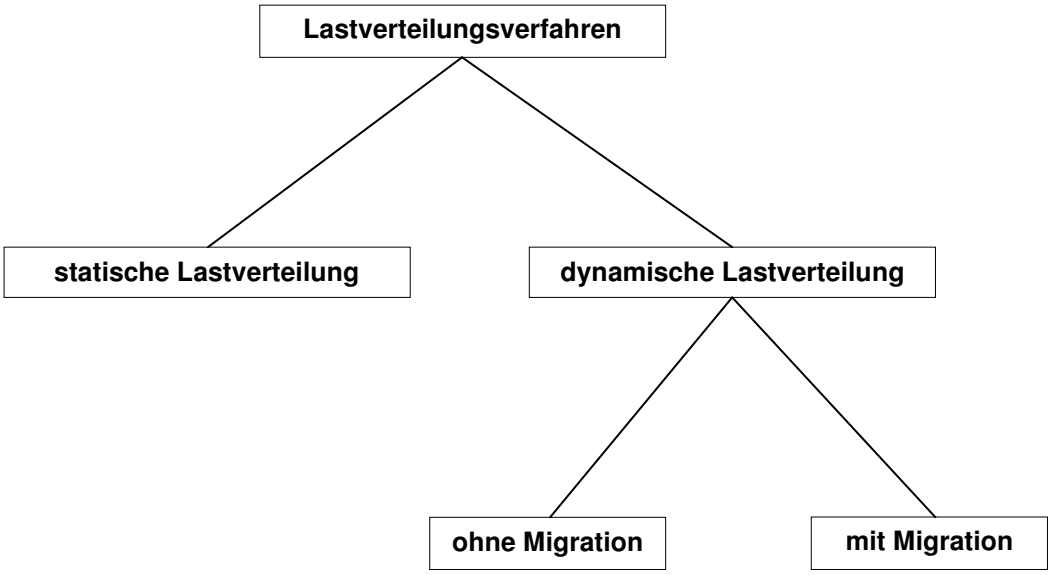
- **Problemstellungen**

  Unabhängig vom verwendeten Lastverbund-Modell ergeben sich zwei zu klärende Fragestellungen :

  - Auf **welcher Workstation** bzw. **welchem Prozessor** (allgemein: **Rechnerknoten**) soll ein **Prozess ausgeführt** werden (**Platzierung** des Prozesses) ?
  - Wie kann eine möglichst **gleichmäßige Verteilung der Rechenlast** auf die verschiedenen Rechnerknoten erreicht werden ?

  Beide Fragestellungen hängen eng zusammen.
  Zu ihrer Lösung wurden zahlreiche **Lastverteilungsverfahren** entwickelt.

- **Überblick über Lastverteilungsverfahren**



◇ **statische Verfahren**

Diese Verfahren verwenden die a-priori über das System und die Prozesse bekannten Parameter.
Keine Berücksichtigung des aktuellen Systemzustands.

◇ **dynamische Verfahren** (adaptive Verfahren)

Diese Verfahren basieren auf dem jeweils aktuell gemessenen Zustand des Systems

◇ **ohne Migration**

Die Entscheidung über die Platzierung eines Prozesses wird zum Zeitpunkt seiner Erzeugung getroffen.
Der Prozess verbleibt während seiner gesamten Lebensdauer auf dem einmal ausgewählten Rechenknoten

◇ **mit Migration**

Ein Prozess kann während seiner Abarbeitung auf einen anderen Rechenknoten verschoben werden.

Lastverteilungsverfahren - statische Lastverteilung

• Allgemeines

◇ Bei einer statischen Lastverteilung werden meist **deterministische Verfahren** angewendet :

▻ Alle benötigten Informationen über das Prozessverhalten liegen vor der Zuteilung vor :
Die Anforderungen der Prozesse an CPU und Speicher, sowie ihr Kommunikationsverhalten (→ Nachrichtenaufkommen zwischen je zwei Prozessen) sind bekannt

▻ Anhand dieser Anforderungen wird jeder Prozess einem bestimmten Rechenknoten zugeteilt.

◇ Es gibt **zahlreiche** auf unterschiedlichen Algorithmen basierende **Verfahren** zur statischen Lastverteilung
(u.a. siehe : Andrew S. Tanenbaum : Verteilte Betriebssysteme, Prentice Hall, 1995)

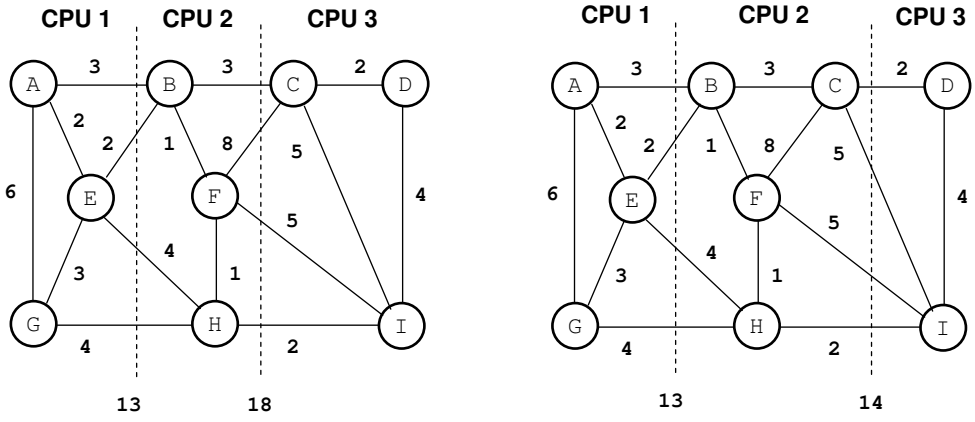
• Beispiel : Graphenbasiertes Verteilungsverfahren

◇ Mit diesem Verfahren können **`n` Prozesse** auf **`k` Rechenknoten** so verteilt werden (`n>k`), dass der **mittlere Kommunikationsaufwand** (Netzwerkverkehr!) zwischen den Rechenknoten **minimal** wird.

Die Prozesse werden mit ihrem Kommunikationsaufwand in einen **gewichteten Graphen** eingetragen :
Jeder **Knoten** entspricht einem **Prozess**, sein Gewicht repräsentiert relevante Aspekte der **Prozessorlast** (z.B. CPU-Belastung, Speicheranforderung usw.)
Jede **Kante** stellt die **Kommunikation** zwischen zwei Prozessen dar. Ihr Gewicht ist ein Maß für den **Kommunikationsaufwand** zwischen den beiden beteiligten Prozessen.

Der Graph muss so in **`k` Untergraphen partitioniert** werden, dass die Summe des an den geschnittenen Kanten auftretenden **Kommunikationsaufwands minimiert** wird, **ohne** dass die einzelnen **Rechenknoten überlastet** werden (d.h. die Summe der innerhalb der Teilgraphen liegenden Knotengewichte einen jeweiligen Schwellenwert nicht überschreitet).

◇ Beispiel : Verteilung von `9` Prozessen (`A .. I`) auf `3` Rechenknoten.
Der Übersichtlichkeit halber sind die Prozessorlasten nicht mit angegeben, die diesbezüglichen Anforderungen seien erfüllt.



Partitionierung 1
Kommunikationsaufwand : `31`

Partitionierung 2
Kommunikationsaufwand : `27`

→ Partitionierung 2 ist günstiger (geringerer Kommunikationsaufwand)

• Problem der statischen Lastverteilung

▻ Die Anforderungsprofile der in den Prozessen realisierten Algorithmen müssen bekannt sein

▻ Dies ist bei immer wiederkehrenden Durchläufen von Algorithmen mit unterschiedlichen Datensätzen gegeben.
z.B. - Wettervorhersage
- Simulationsverfahren

Lastverteilungsverfahren - dynamische Lastverteilung (1)

• Allgemeines

- ausschlaggebend für die Zuteilung von Aufträgen ist die **momentane Auslastung** der Rechenknoten
- ein zusätzlich bekanntes Anforderungsprofil der verschiedenen Aufträge kann hilfreich sein.

• Lastmessung

- Die Kenntnis der jeweils aktuellen Auslastung der Rechenknoten erfordert eine ständige Messung der Last
  → **Lastmessung** ist eine **zentrale Komponente** der dynamischen Lastverteilung

- Für die Bewertung der Rechenlast werden herangezogen :
  - **Prozessorauslastung**
  - **Speicherauslastung**
  - **Kommunikationslast**

- **Metriken für die Prozessorauslastung**:
  - Gesamtanzahl der Prozesse.
  - Anzahl der Prozesse, die sich im Zustand aktiv (*running)* oder bereit (*ready)* befinden.
  - Zeitanteil, den der Prozessor im letzten vergangenen Zeitintervall im idle-Prozess verbracht hat
  - Mittlere Bedienzeit der Prozesse.
    Diese ergibt sich aus der mittleren Verweildauer eines Prozesses im Zustand *ready*.

- **Metriken für die Speicherauslastung** :
  - Anzahl der freien Speicherseiten im Arbeitsspeicher des Rechners.
  - Anzahl der freien Speicherseiten im virtuellen Speicher d.h. auf den Festplatten des Rechners.

- **Metriken für die Kommunikationslast**:
  - Anzahl der gesendeten und empfangenen Nachrichten pro Zeiteinheit.
  - Anzahl der gesendeten und empfangenen Bytes pro Zeiteinheit.

- Die jeweils benötigten Auslastungs-Messwerte müssen von **Monitorprogrammen** während der Laufzeit, d. h. **dynamisch**, für jeden Rechenknoten ermittelt und gespeichert werden.

- Die Monitorprogramme müssen möglichst **einfach** und **wenig rechenintensiv** sein, damit der durch sie verursachte **Belastungs-Overhead** die tatsächlichen Verhältnisse nicht zu sehr verfälscht.

- Die Auswertung der über die einzelnen Rechner verfügbaren Daten entspricht der Lösung eines **Optimierungs-problems**.
  Um die Komplexität der Lösungs-Algorithmen möglichst gering zu halten, werden häufig **Heuristiken** in diese einbezogen.

Lastverteilungsverfahren - dynamische Lastverteilung (2)

- **Verteilung der Lastinformation**

  ◇ **Client-initiiert**

  ▻ Ein Client , der einen Rechenauftrag zu vergeben hat, erfragt – mittels Broadcast oder Multicast -
  - bei allen Rechnerknoten,
  - bei einer Teilmenge der Knoten
  - oder bei einem speziellen Knoten

  die Lastinformation ab.

  ▻ Er bewertet die erhaltene Information, um den Rechnerknoten mit der aktuell geringsten Belastung zu finden.
  Diesen wählt er aus.

  ▻ Diese Strategie wird auch **Lastabstoßung** genannt.

  ◇ **Server-initiiert**

  ▻ Ein Rechnerknoten (Server), der sich selbst unterbeschäftigt fühlt, erfragt die Lastinformationen von anderen Rechnerknoten (den möglichen Clients) ab.

  ▻ Die Bewertung nimmt der Server vor.
  Er wählt den am meisten belasteten Rechnerknoten aus.

  ▻ Diese Strategie wird auch als **Lastanziehung** bezeichnet.

  ▻ Besonders geeignet, wenn Prozessmigration stattfinden soll

  ◇ **Periodisch Server-initiiert**

  ▻ Alle als Server in Frage kommenden Rechnerknoten schicken periodisch - mittels Broadcast oder Multicast zu allen anderen Knoten oder zu einer Teilmenge der Knoten ihre Lastinformation.

  ▻ Die Bewertung erfolgt entweder lokal unabhängig bei jedem Empfänger oder durch einen über die Server verteilten Bewertungsalgorithmus

  ▻ Hier kann sowohl Lastanziehung als auch Lastabstoßung gemischt realisiert werden.

  ◇ In allen Varianten kann die Information auch bei einer **zentralen Verwaltungsinstanz** zur Verfügung gehalten werden.

  ◇ **Wichtiger Aspekt**:

  ▻ Durch die **zeitliche Verzögerung** der Informationsverteilung und die **Dauer ihrer Bewertung** kann die Lastinformation **nie den aktuellsten Zustand** widerspiegeln.

  ▻ Die Bewertung kann damit zu falschen, überholten Ergebnissen kommen.

Lastverteilungsverfahren - dynamische Lastverteilung (3)

- **Architektur eines dynamischen Lastverteilungssystems**

  Dynamische Lastverteilungssysteme bestehen typischerweise aus den folgenden 5 Komponenten:

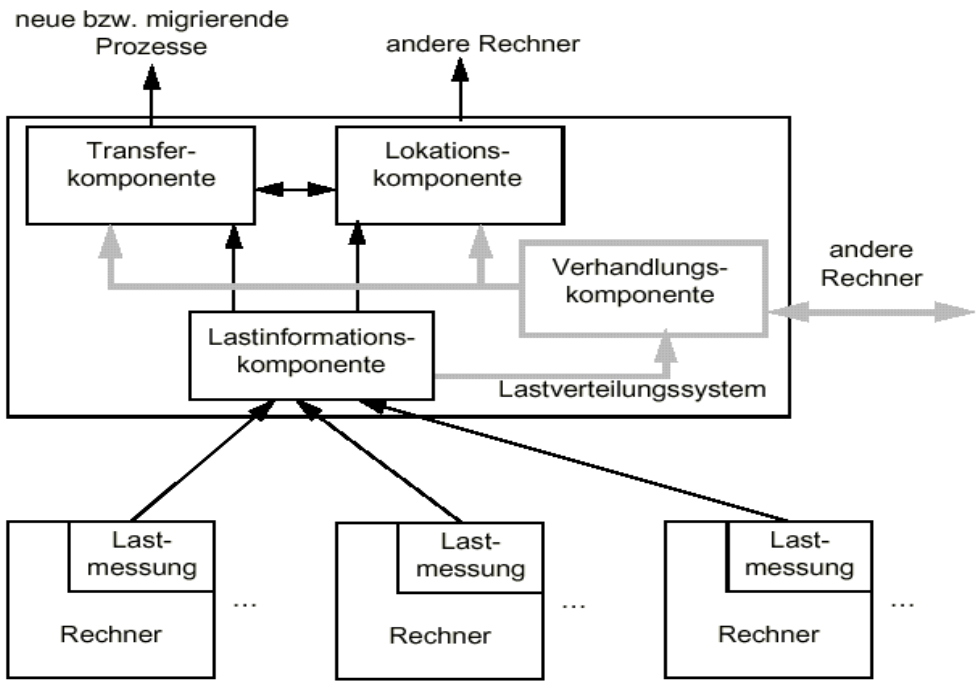
  ◇ **Monitor zur Lastmessung**
  - ▻ befindet sich auf jedem Rechnerknoten
  - ▻ dient zur Messung der lokal vorhandenen Last.

  ◇ **Lastinformationskomponente**
  - ▻ sammelt die mittels den Monitoren gewonnenen Lastinformationen
  - ▻ und bereitet diese auf für eine Verwendung durch die Transfer- und die Lokationskomponente

  ◇ **Transferkomponente**
  - ▻ wählt die auszulagernden Prozesse aus
  - ▻ sorgt für eine Anpassung der entfernten Umgebung
  - ▻ und veranlasst die Übertragung von Code und Daten

  ◇ **Lokationskomponente**
  - ▻ wählt den Wirts-Rechenknoten aus

  ◇ **Verhandlungskomponente**
  - ▻ nur bei verteilter Implementierung des Lastverteilungssystems vorhanden
  - ▻ sorgt für den Austausch der Lastinformationen zwischen den einzelnen – lokalen – Lastverteilungssystemen
  - ▻ und übernimmt die Verhandlung zur Lastverteilung und die gegenseitige Abstimmung zwischen den einzelnen – lokalen – Lastverteilungssystemen



Prinzipielle Struktur eines Lastverteilungssystems
(aus : M. Weber, Verteilte Systeme, 1998)

Dynamische Lastverteilung ohne Migration (1)

- **Allgemeines**

  - ▻ **Ziel** der dynamischen Lastverteilung ohne Migration ist es, **neu entstehende Prozesse** anhand der **aktuellen Lastbewertung** auf dem "**richtigen**" **Rechenknoten** (Server) zu erzeugen.
  - ▻ Man nennt diese Aufgabe auch ***initial placement***.
  - ▻ Die **Steuerung** der dynamischen Lastverteilung kann sowohl vom initiierenden Rechenknoten (Client) als auch vom aufnehmenden Rechenknoten (Server) erfolgen.

- **Client-gesteuerte Lastverteilung**

  - ◇ **Prinzip**:
    - ▻ Der **Client** selbst **such**t einen geeigneten **Server** zur Ausführung seines Auftrags.
    - ▻ Das **Lastverteilungssystem** ist – bis auf den Monitor zur Lastmessung – **vollständig beim Client** realisiert.
    - ▻ Es verfügt über **keine Verhandlungskomponente**.

  - ◇ **Grundlegender Algorithmus**:

```
Broadcast an alle Rechenserver :
  "Bitte Lastinformation abliefern";
Erwarten Rückantwort von allen Servern;
Ermittlung des optimalen Servers;
CreateProcess(Code, Umgebung) an optimalen Server senden;
Erwarten Antwort mit Prozessidentifikation;
```

  - ◇ **Probleme**:
    - ▻ Die **Lastinformation** kann **veraltet** und damit **unzuverlässig** sein, insbesondere, wenn die Zeitdauern bis zur Rückantwort an den anfordernden Client eine große Streuung aufweisen.
      → Bei der Bewertung der Lastinformation ist das entsprechend zu berücksichtigen
    - ▻ Die gleichzeitige Anfrage mehrerer Clients provoziert **Überlastung einzelner Server**.
      → Da alle Clients die gleiche Lastinformation erhalten, wählen sie möglicherweise alle denselben Server
    - ▻ Wegen der möglicherweise großen Zahl von Broadcasts ist das Verfahren **schlecht skalierbar**

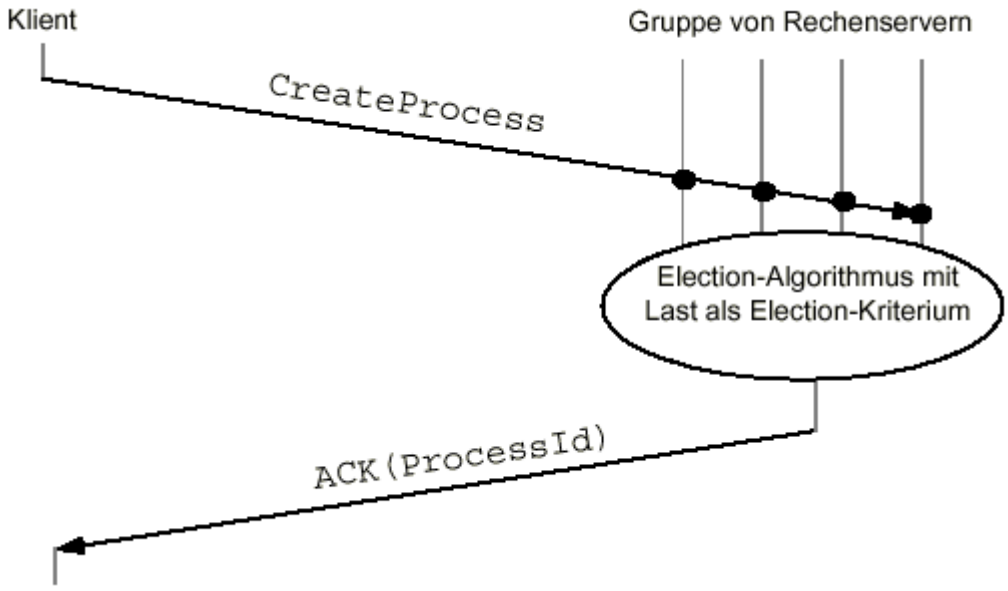
  - ◇ In der Praxis lassen sich **gute Ergebnisse** bereits mit relativ **einfachen Auswahlverfahren** erzielen, z. B :
    - ▻ Verfahren 1 :
      Der Server, an den der `CreateProcess`-Aufruf geschickt wird, wird **zufällig ausgewählt**.
      Falls dieser Server überlastet ist, schickt er den Aufruf nach dem gleichen Prinzip an einen anderen Server weiter , usw.
    - ▻ Verfahren 2 :
      Die Auswahl des optimalen Servers erfolgt aus einer zufällig ausgewählten **kleinen Teilmenge aller Server**.

Dynamische Lastverteilung ohne Migration (2)

- **Server-gesteuerte Lastverteilung**

  ◈ Variante 1 : **Servergruppe mit Election**

  ▻ Die Server sind in Gruppen zusammengefasst.
  ▻ Ein Client sendet per Multicast eine `CreateProcess`-Anforderung an alle Server einer Gruppe.
  ▻ Die Server starten daraufhin einen Election-Algorithmus mit der aktuellen Last der Server als Auswahlkriterium.
  ▻ Der Server mit der geringsten Last erzeugt den Prozess
  ▻ und sendet an den Client eine Bestätigungsmeldung mit der Prozess-ID des erzeugten Prozesses.



Server-gesteuerte Lastverteilung mit Election
(aus M. Weber, Verteilte Systeme, 19998)

▻ **Aufteilung des Lastverteilungssystems** :
  - Alle Komponenten – außer der Transferkomponente – sind auf den Servern implementiert.
  - Die Lokations- und Verhandlungskomponente werden dabei durch den Election-Algorithmus realisiert.

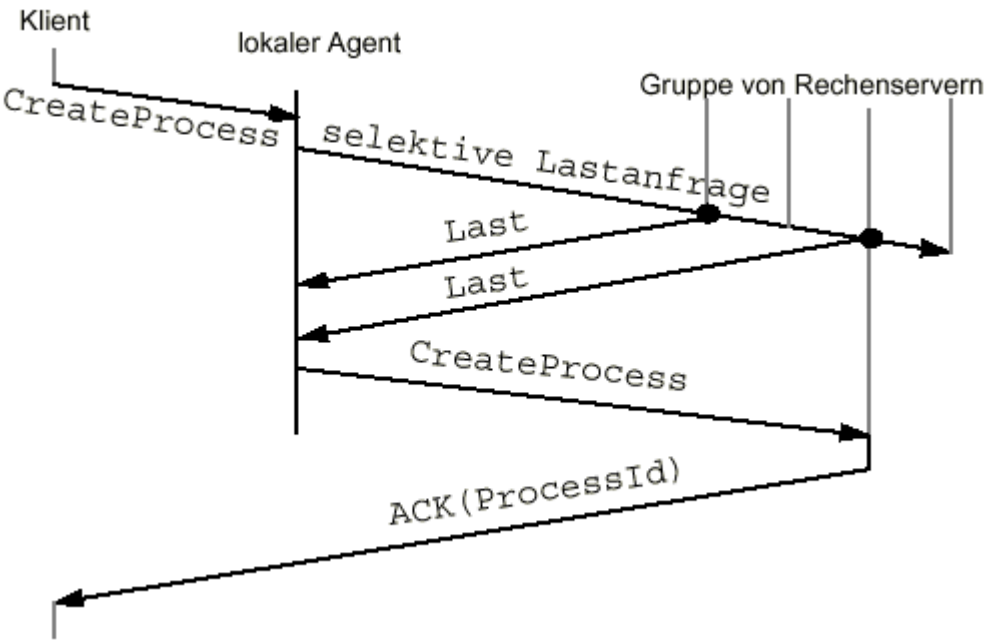
▻ Das Verfahren arbeitet bei kleinen Server-Gruppen gut
Bei größeren Servergruppen wird der Election-Algorithmus zum kritischen Punkt

Dynamische Lastverteilung ohne Migration (3)

- **Server-gesteuerte Lastverteilung**, Forts.

  - Variante 2 : **Delegation an einen Agenten**

    - Ein Client stellt eine `CreateProcess`-Anfrage an einen **Agenten**.
    - Der Agent stellt eine **Lastanfrage** an einen **Teil der Server**.
    - Aus den von den Servern erhaltenen Lastinformationen wählt er den Server, der den Auftrag bekommt, aus
    - Da der Agent aufgrund vorheriger Anfragen bereits über ein gewisses Bild der Lastverteilung verfügt, kann er seine **Lastanfragen gezielt** an bestimmte Server senden



Server-gesteuerte Lastverteilung mit Delegation an einen Agenten
(aus M. Weber, Verteilte Systeme, 1998)

- **Aufteilung des Lastverteilungssystems** :
  - Der Agent vereint die Lastinformations-, Lokations- und Transferkomponente.
  - Die Monitore zur Lastmessung befinden sich in den Servern.
  - Falls mehrere Agenten existieren, die untereinander Informationen austauschen, verfügen sie zusätzlich über Verhandlungskomponenten.

- Durch die selektiven Anfragen des Agenten **skaliert** das Verfahren **gut**.

Dynamische Lastverteilung mit Migration (1)

- **Allgemeines**

  - ▻ **Ziel** der dynamischen Lastverteilung mit Migration ist der **dynamische Lastausgleich** während der Laufzeit.
    → laufende Prozesse werden von einem stark belasteten Rechenknoten auf einen gering belasteten Knoten verlagert.
  - ▻ **Neue Prozesse** werden erst **lokal erzeugt** und dann gegebenenfalls **ausgelagert**
  - ▻ Die dynamische Lastverteilung mit Migration ist **symmetrisch** angelegt :
    → Jeder Rechenknoten kann sowohl Client als auch Server sein.
    → Alle Komponenten des Lastverteilungssystems sind auf allen Rechenknoten vorhanden.

- **Prinzipien :**

  - ◇ **Lastabstoßung**
    - ▻ In einem Rechenknoten wird ein **Lastschwellwert überschritten**.
    - ▻ Der Knoten **sucht** nach einem anderen **weniger belasteten** Rechenknoten
    - ▻ und **verlagert Last** zu diesem.
    - ▻ **Problem** : Wenn das **Gesamtsystem überlastet** wird.
      In diesem Fall führt Lastabstoßung zu **Trashing**:
      → Alle Rechner sind überwiegend nur noch damit beschäftigt, Last abzustoßen
  - ◇ **Lastanziehung**
    - ▻ In einem Rechenknoten wird ein **Lastschwellwert unterschritten**
    - ▻ Der Knoten **bietet** seine **Ressourcen** anderen Rechenknoten **zur Nutzung an**.
    - ▻ **Stark belastete Knoten** können **Prozesse** zu dem unterbelasteten Knoten **verlagern**.
    - ▻ Damit **Trashing vermieden** wird, muss der Ziel-Knoten der **Verlagerung zustimmen**

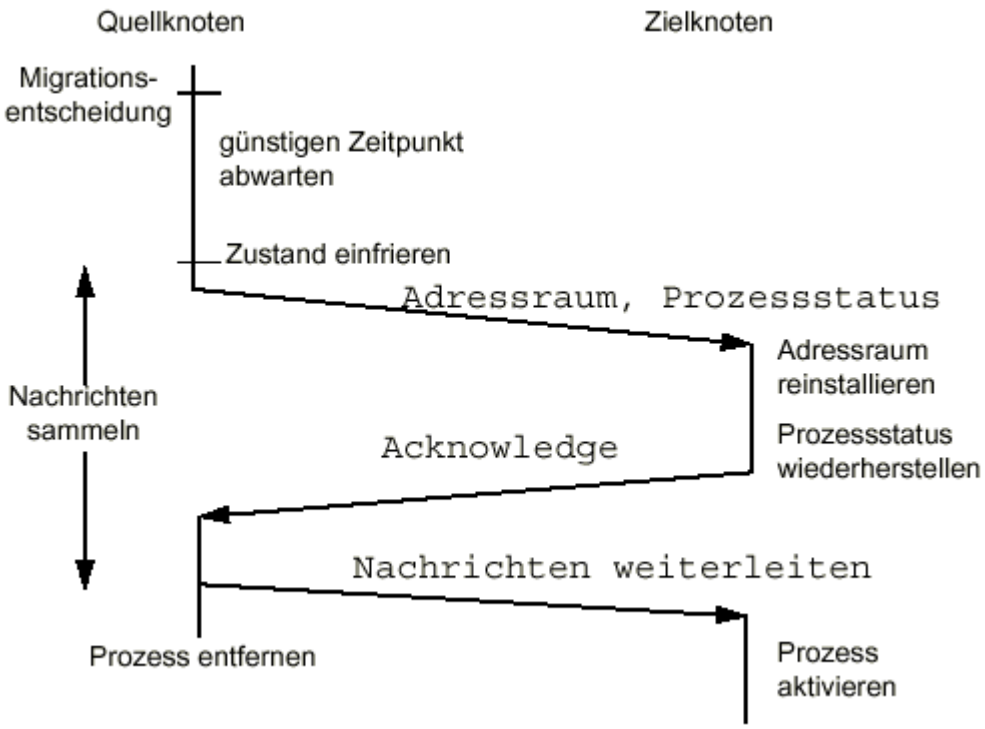
- **Grundbedingung für eine Migration**

  - ◇ Für eine erfolgreiche Migration eines laufenden Prozesses muss für den Prozess auf dem **Ziel-Rechenknoten** genau **derselbe Zustand**, wie er auf dem Quellknoten vorlag, wiederhergestellt werden.
    → Der migrierende Prozess muss auf dem Zielknoten exakt die **gleiche aktuelle Umgebung** wie auf dem Quellknoten vorfinden
    - ▻ Hierfür muss der **aktuelle Adressraum** des Prozesses und sein **sonstiger Zustand erfasst** und auf den Zielknoten **übertragen** werden
    - ▻ Außerdem müssen alle **Kommunikationsbeziehungen** des migrierenden Prozess **erhalten** bleiben.

Dynamische Lastverteilung mit Migration (2)

- **Ablauf einer Migration**

  - Nachdem die **Entscheidung zur Migration** eines Prozesses getroffen worden ist, muss ein **günstiger Zeitpunkt** für die Migration abgewartet werden.
  - Dieser ist beispielsweise dann erreicht, wenn der Prozess infolge eines Betriebssystemaufrufs in den **Wartezustand** (`waiting, blocked`) übergegangen ist.
  - Der zu diesem Zeitpunkt vorliegende **Zustand** wird **eingefroren**
  - **Adressraum** und **Prozessstatus** werden zum Ziel-Rechenknoten **übertragen** und dort **reinstalliert**.
  - Der Prozess wird auf dem Ziel-Rechenknoten **reaktiviert** und fortgesetzt



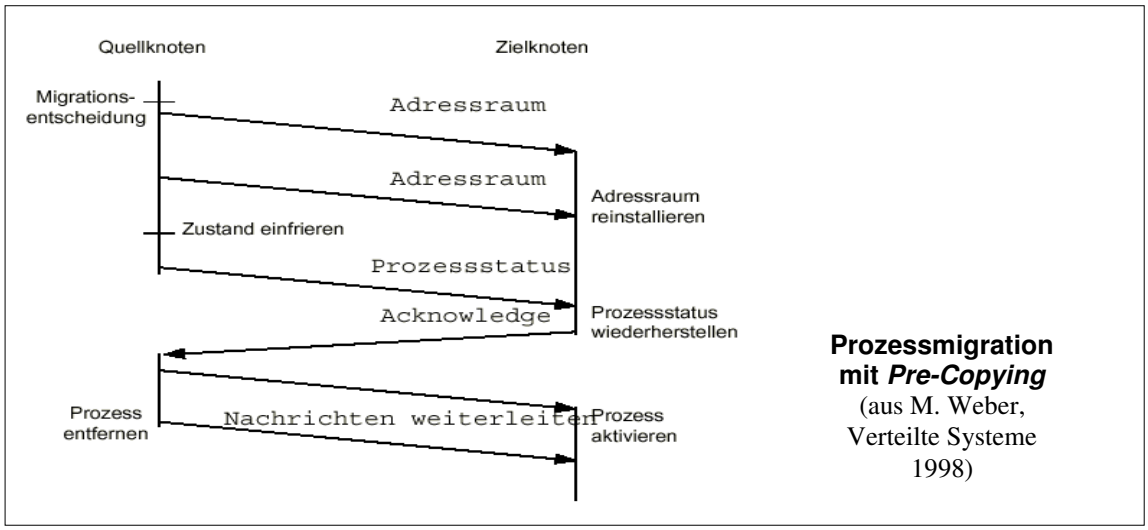
Ablauf einer Prozessmigration
(aus M. Weber, Verteilte Systeme, 1998)

  - **Während** der **Migrationsphase** für den migrierenden Prozess **eintreffende Nachrichten** muss der Quellknoten **aufbewahren**.
  - Nach der Reaktivierung des Prozesses im Zielknoten muss der Quellknoten diesem die **Nachrichten zuleiten**.
  - Die vom Prozess unterhaltenen **Kommunikationskanäle** müssen nach der Migration entweder auf den neuen **umgeleitet** werden
  - oder es **verbleibt** ein **Durchreicheprozess** (Stellvertreter, Proxy) auf dem alten Knoten.
  - Auf dem **alten Knoten** muss der **Prozess** aus der Prozesstabelle **entfernt** werden.

Dynamische Lastverteilung mit Migration (3)

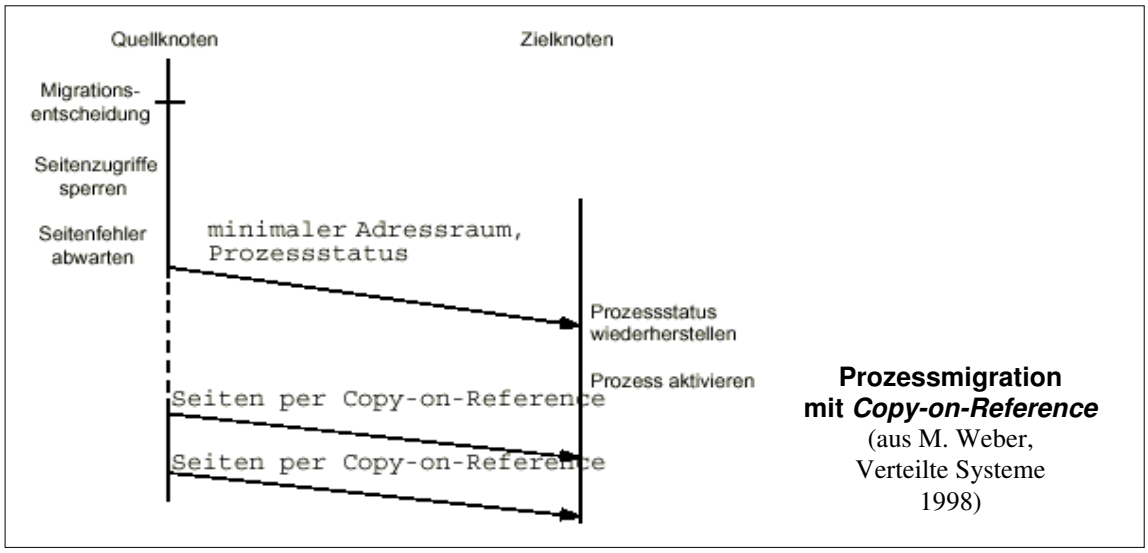
- **Effizientere Varianten des Migrationsablaufs**

  - ***Pre-Copying***
    - Mit der Übertragung des Adressraums wird bereits zwischen der Migrationsentscheidung und dem Einfrieren des Prozesses begonnen.
    - Nach dem Einfrieren müssen die bis dahin neu bearbeiteten Seiten aktualisiert werden.



Prozessmigration mit *Pre-Copying*
(aus M. Weber, Verteilte Systeme 1998)

  - ***Copy-on-Reference* (*Post-Copying*)**
    - Nach der Migrationsentscheidung werden alle **Seitenzugriffe** des Prozesses **gesperrt**.
    - Der erste Zugriff zu einer gesperrten Seite verursacht einen **Seitenfehler** (*Page Fault*)
    - Die aktuell benötigte Seite bildet einen **minimalen Arbeitsadressraum**, der zusammen mit dem Prozessstatus **zum Zielknoten übertragen** wird.
    - Anschließend wird der **Prozess** auf dem **Zielknoten reaktiviert**.
    - Zugriffe zu **noch nicht übertragenen Seiten** bewirken **Seitenfehler** im Zielknoten.
      Die benötigten Seiten müssen vom Quellknoten **nachgeliefert** werden.



Prozessmigration mit *Copy-on-Reference*
(aus M. Weber, Verteilte Systeme 1998)