Prozessorvergabe - vert. Syst.: Modelle

6 Prozessorvergabe in verteilten Systeme

6.1 Systemmodelle

- **Arbeitsplatzrechner-Modell**

  Das System besteht aus einer Menge von vernetzten Arbeitsplatzrechnern, zu denen jeder Benutzer Zugang hat

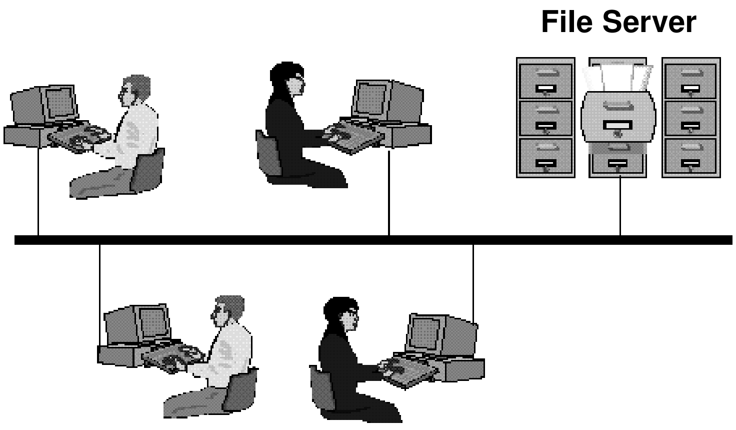
- **Prozessorpool-Modell**

  Benutzer haben Zugang zu einem einfachen Terminal (X-Terminal), ihre Aufträge werden zur Ausführung an einen Pool von Rechnern gesandt

- **Hybrides Modell**

Prozessorvergabe - vert. Syst.: Modelle

- **Arbeitsplatzrechner-Modell**



- Berechnungen werden am lokalen Rechner durchgeführt
- Dateien werden am *File Server* gespeichert
- Lokaler Plattenspeicher wird für Paging, Swapping, temporäre Dateien un Dateipufferung benutzt
- Schlechte Betriebsmittelnutzung

BP 2

Prozessorvergabe - vert. Syst.: Modelle

Nutzung freier Kapazitäten

Was ist ein freier Arbeitsplatzrechner?

- Ein Rechner, an dem seit mehreren Minuten keine Eingabe erfolgte und kein Benutzerprozeß läuft

Wie findet man freie Arbeitsplatzrechner?

- Server-initiiert: Ein freier Rechner macht bekannt, daß er frei geworden ist
- Client-initiiert: Benutzerrechner sucht per Broadcast nach einem freien Rechner

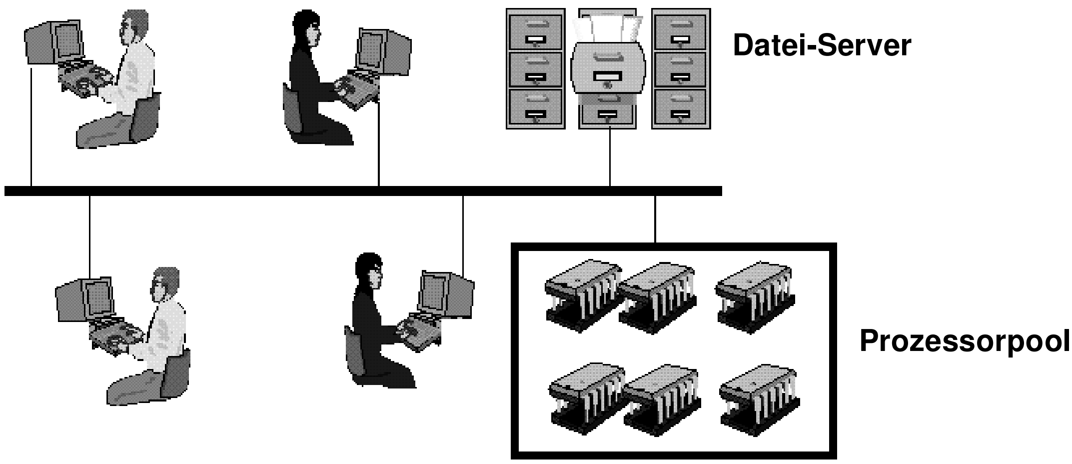
Was geschieht, wenn der Besitzer seinen Rechner benötigt

- **Nichts:** Der Benutzer findet einen verlangsamten Rechner vor; keine gute Idee!
- **Verlagerung auf anderen freien Rechner:** Erheblicher Aufwand
- **Fernen Prozeß mit niedrigster Priorität weiter bearbeiten:** Nicht bemerkbar bezüglich Rechenleistung, belegt aber Betriebsmittel.

BP 2

Prozessorvergabe - vert. Syst.: Modelle

Prozessorpool-Modell

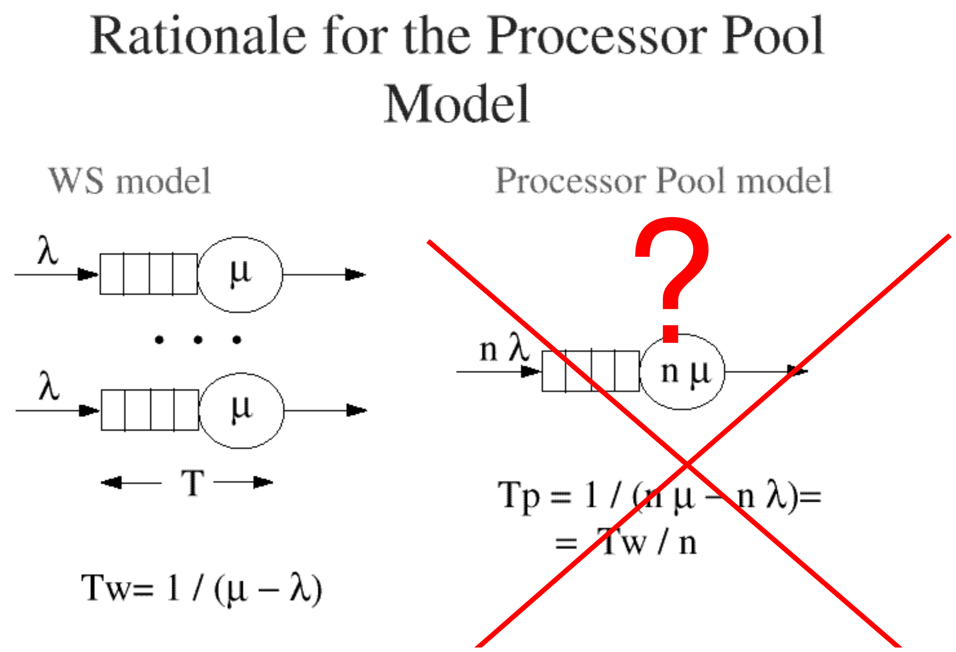


- **Prozessoren werden dynamisch nach Bedarf zugeteilt**

Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann
Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

Prozessorvergabe - vert. Syst.: Modelle

Nicht alles, was man im Internet findet, muß richtig sein!



Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

Prozessorvergabe - vert. Syst.: Modelle

- **Rechnerzuteilung**
  - **Ziel: Auf welchem Prozessor sollte ein Prozeß laufen um Lastausgleich zu erreiche und die Ausführungszeit zu optimieren?**
  - **Annahmen**
    - **Alle Prozessoren sind binär-kompatibel (evtl. mit unterschiedlicher Ausführungsgeschwindigkeit)**
    - **Das System ist voll vermascht, d. h. jeder Prozessor kann mit jedem anderen Nachrichten austauschen**
  - **Quantifizierung des Vorteils von Lastausgleich**

Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

BP 2 Prozessorvergabe - vert. Syst.: Modelle

- **Man betrachte eine Menge von M/M/1-Systemen**
  - **Es sei p die Wahrscheinlichkeit, daß wenigstens ein Server frei ist und wenigstens ein Auftrag auf Bearbeitung wartet**
  - **Zunächst Betrachtung eines Arbeitsplatzrechners**
    - $p_0 = 1 - \rho$ Wahrscheinlichkeit, daß er frei ist
    - $p_1 = \rho p_0 = (1 - p_0)p_0$ Wahrscheinlichkeit, daß er genau einen Auftrag enthält
  - **Menge gleichartiger Prozessoren**
    - $q_i = p_0^i$ Wahrscheinlichkeit, daß i vorgegebene Prozessoren frei sind
    - h_{N-i} Wahrscheinlichkeit, daß $N - i$ vorgegebene Prozessoren beschäftigt sind und wenigsten eine ihrer Warteschlangen nicht leer ist
    - h_{N-i} = {Wahrscheinlichkeit, daß alle $N - i$ wenigstens einen Auftrag enthalten} - {Wahrscheinlichkeit, daß alle genau einen Auftrag enthalten}

$$\Rightarrow \quad h_{N-i} = (1 - p_0)^{N-i} - ((1 - p_0)p_0)^{N-i}$$

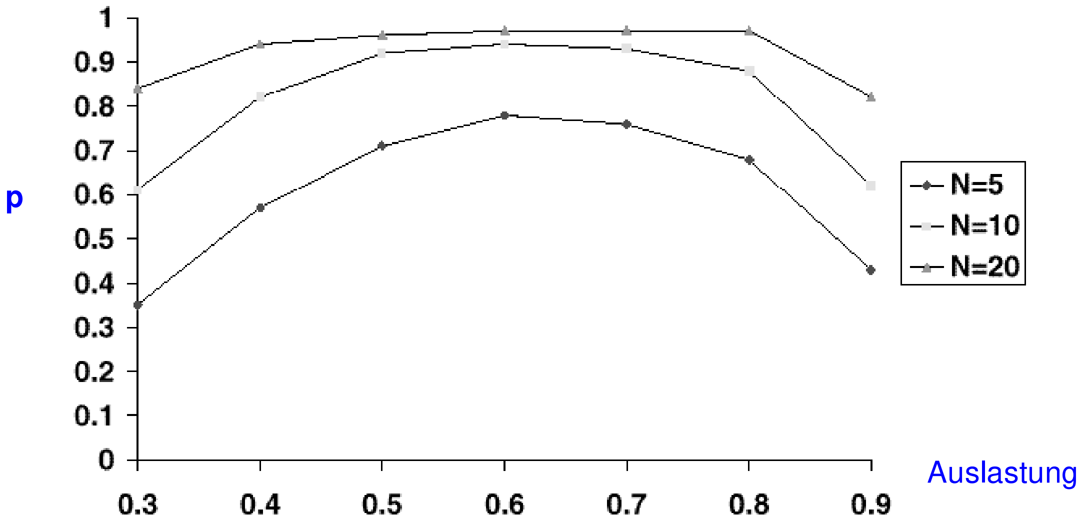
BP 2 Prozessorvergabe - vert. Syst.: Modelle

Also
$$p = \sum_{i=1}^{N} \binom{N}{i} q_i h_{N-i}$$
$$= \sum_{i=1}^{N} \binom{N}{i} p_0^i \{(1 - p_0) - [(1 - p_0)p_0]^{N-i}\}$$
$$= 1 - (1 - p_0)^N (1 - p_0^N) - p_0^N (2 - p_0)^N$$

BP 2

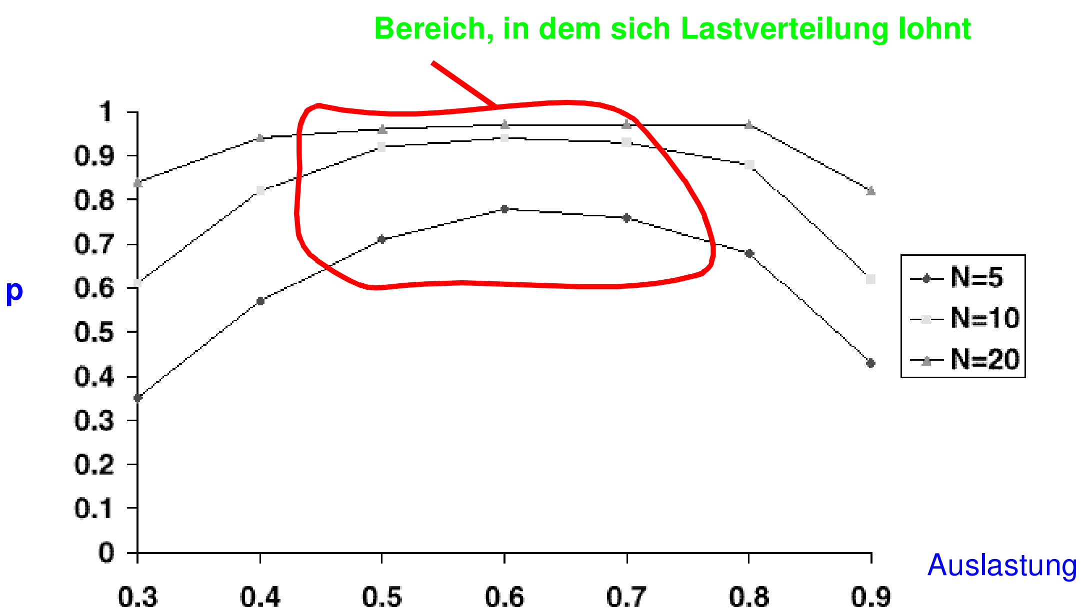
Prozessorvergabe - vert. Syst.: Modelle

- **Wahrscheinlichkeit p, daß wenigstens ein Prozessor frei und ein Auftrag wartend ist, wenn ohne Lastausgleich gearbeitet wir**



BP 2

Prozessorvergabe - vert. Syst.: Modelle



Prozessorvergabe - vert. Syst.: Modelle

❑ **Lastverteilung versus Lastausgleich**

- **Lastverteilung (*load sharing*)**
  - Es wird versucht zu vermeiden, daß ein Rechner leersteht, während in irgendeiner Warteschlange ein Auftrag wartet
- **Lastausgleich (*load balancing*)**
  - Es wird versucht, die Last gleichmäßig auf alle Rechner zu verteilen. Größerer Overhead durchTransfers kann den erwarteten Vorteil in einen Nachteil verkehren.

Prozessorvergabe - vert. Syst.: Lastverteilun

6.2 Lastverteilung

❑ **Wesentliche Gesichtspunkte**

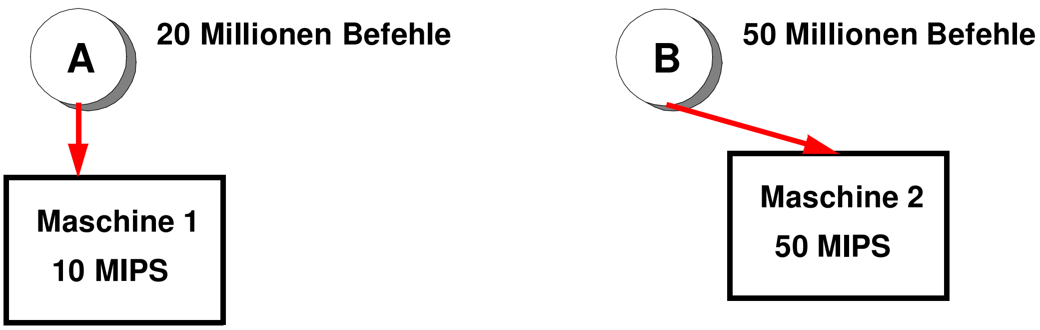
- **Die Warteschlangenlänge korreliert sehr gut mit der Verweilzeit**
- **Bei großen Übertragungsverzögerungen sollten Veränderungen der Wartschlangenlänge verbreitet werden, wenn Transferentscheidungen getroffen werden, nicht wenn neue Prozesse ins System komme**
- **Timeouts werden zur Reduktion der Warteschlangenlänge benutzt, wenn Prozeß nicht ankommt**
- **Auch die Auslastung kann in die Entscheidungen einbezogen werden**

BP 2 Prozessorvergabe - vert. Syst.: Gesichtspunkte

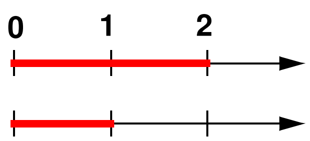
- **Rechnerzuordnung**
  - **Strategien**
    - Nicht-migrierend (*nonmigratory*): Ein Prozeß kann nach seinem Start nicht mehr verlagert werden
    - Migrierend (*migratory*): Prozesse können während ihrer Ausführung verlagert werden um Lastausgleich zu erzielen
  - **Optimierungsziele**
    - Maximierung der Gesamtauslastung
    - Minimierung der mittleren Verweilzeit
    - Minimierung des Verhältnisses von Warte- zu Bedienzeit

BP 2 Prozessorvergabe - vert. Syst.: Gesichtspunkte

Beispiel

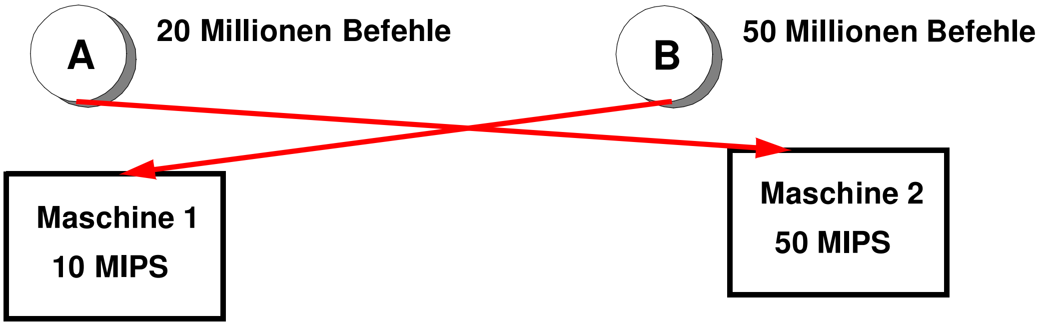


U_A = 20/10 sec = 2 sec
U_B = 50/50 sec = 1 sec
$E[U]$ = (2+1)/2 = 1.5 sec
ρ_1 = 2/2 = 100%
ρ_2 = 1/2 = 50%
$E[\rho]$ = 3/4 = 75%

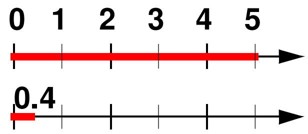


Prozessorvergabe - vert. Syst.: Gesichtspunkte

I



U_A = 20/50 sec = 0.4 sec
U_B = 50/10 sec = 5 sec
E[U] = (5+0.4)/2 = 2.7 sec
ρ_1 = 5/5 = 100%
ρ_2 = 0.4/5 = 8%
E[ρ] = (5+0.4)/10 = 54



Prozessorvergabe - vert. Syst.: Lastverteilungsalgorithmen

6.3 Komponenten eines Lastverteilungsalgorithmus

- **Transport-Strategie (*Transfer Policy*)**

  Wann soll ein Prozeß verlagert werden?

- **Auswahl-Strategie (*Selection Policy*)**

  Welcher Prozeß soll verlagert werden?

- **Plazierungsstrategie (*Location Policy*)**

  Wohin soll der Prozeß verlagert werden

- **Informations-Strategie (*Information Policy*)**

  Welche Informationen über andere Rechner werden wann und woher eingesammelt?

BP 2 Prozessorvergabe - vert. Syst.: Lastverteilungsalgorithmen

- **Transportstrategie**
  - **Schwelle (*threshold*)**
    - Wenn die vorhandene Last mehr als T Zeiteinheiten für ihre Abarbeitung benötigt, werden Prozesse ausgelagert.
    - Wenn die Last unter T Zeiteinheiten zurückgeht, werden Prozesse eingelagert
  - **Verfeinerun**
    - Unterer und oberer Schwellwert zur Vermeidung von Prozeßflattern
- **Auswahl-Strategie**
    - Auswahl eines neu hinzugekommen Prozesses; einfach zu migrieren
    - Auswahl nach Verlagerungsaufwand (z. B. möglichst wenig lokaler Kontext)
    - Prozeß verlagern, wenn dadurch seine Verweilzeit verkürzt werden kann

BP 2 Prozessorvergabe - vert. Syst.: Lastverteilungsalgorithmen

- **Plazierungs-Strategie**
  - **Pollen: Andere Rechner werden regelmäßig auf ihre Aufnahmefähigkeit hin überprüft**
    - Überprüfung seriell oder parallel
    - Zufällige Auswahl
    - Nächster Nachbar
    - Aufgrund von früheren Informationen
  - **Anfrage per Broadcast**

Prozessorvergabe - vert. Syst.: Lastverteilungsalgorithmen

Informations-Strategie

- Auf Anforderung (*demand-driven*): **Informationen werden nur eingesammelt, wenn der Rechner in eine Verlagerung involviert wir**
  - Sender-initiiert (*sender-initiated*): Sender sucht Empfänger
  - Empfänger-initiiert (*receiver-initiated*): Empfänger sucht Prozeß
  - Symmetrisch initiiert (*symmetrically-initiated*): Kombination der beiden vorangehenden Methoden
- Periodisch (*periodic*): **Rechner tauschen regelmäßig Informationen aus**
  - Ständiger Overhead
  - Nicht an Systemlast anpaßbar: Bei Hoch- und Niederlast nutzlose Zusatzbelastung
- Durch Zustandsänderungen veranlaßt (*state-change-driven*): **Rechner verbreiten Information nur, wenn sich ihr Zustand ändert**
  - Statusinformation wird verbreitet, nicht eingesammelt
  - zentralisiert: Statusinformationen werden zentral verwaltet und ausgewertet
  - verteilt: Statusinformationen gehen an alle Rechner
  - Zusatzaufwand für Entgegennahme der Statusinformationen ist Funktion der Systemlast

Prozessorvergabe - vert. Syst.: Lastverteilungsalgorithmen

Stabilität und Effektivität

- **Algorithmus ist** instabil (*unstable*), **wenn die gesamte Ankunftsrate von Last die Leistungsfähigkeit des Rechners überfordert**
- **Algorithmus ist** instabil (*unstable*), **wenn mit einer endlichen Wahrscheinlichkeit Prozesse von einem Rechner zum anderen wandern oder Fortschritt zu erziele**
- **Ein Algorithmus ist** effektiv (*effective*) **, wenn er die Systemleistung erhöht**

BP 2

Prozessorvergabe - vert. Syst.: Lastverteilungsalgorithmen

❑ **Klassifikation von Lastverteilungsalgorithmen**

- **Dynamisch: Entscheidung auf Grund der Knotenlast**
  - Ausnutzen kurzfristiger Lastschwankungen
  - Zusatzbelastung beim Sammeln, Speichern und Auswerten von Zustandsinformation
- **Statisch: Transferentscheidungen auf Grund von a-priori-Informatio**
- **Adaptiv: Wie dynamisch, aber zusätzlich adaptive Regelalgorithmen**

BP 2

Prozessorvergabe - vert. Syst.: Lastverteilungsalgorithmen

◆ **Sender-initiierte Algorithmen**

- **Transport-Strategie:**
  - Schwellwert-Strategie, die auf der Länge der Bereitwarteschlange basiert
- **Auswahl-Strategie: Neu angekommene Prozesse**
- **Plazierungs-Strategie:**
  - Zufällig: Vermeidung von Flattern durch Begrenzung der Transfers
  - Schwellwert-basiert: Befragen einer durch den Schwellwert begrenzten Menge von Knoten, um einen Empfänger zu finden. Falls kein Kandidat gefunden wird, lokal bearbeiten
  - Kürzeste Warteschlange: Befragen einer begrenzten Menge von Knoten und Auswahl desjenigen, mit der kürzesten Warteschlange
- **Informations-Strategie:**
  - Einsammeln der Zustandsinformation wird ausgelöst, wenn ein Knoten Last abgeben möchte
- **Stabilität:**
  - Instabil bei hoher Systemlast, da mit großer Wahrscheinlichkeit kein Empfänger gefunden wird und damit nur Zusatzaufwand entsteht

Prozessorvergabe - vert. Syst.: Lastverteilungsalgorithmen

- **Empfänger-initiierte Algorithme**
  - **Transport-Strategie:**
    - Schwellwert-Strategie, die auf der Länge der Bereitwarteschlange basiert
  - **Auswahl-Strategie: Irgendeine**
  - **Plazierungs-Strategie:**
    - Zufällig: Befragung einer begrenzten Menge von Knoten, ob ihr Schwellwert überschritten ist
    - Falls Suche fehlschlägt, warten bis ein anderer Prozeß fertig wird, dann erneut überprüfen (evtl. auch nach einem vorbestimmten Zeitintervall)
  - **Informations-Strategie:**
    - Einsammeln der Zustandsinformation wird ausgelöst, wenn ein Knoten zusätzliche Last aufnehmen kann
  - **Stabilität:**
    - Stabil. Bei hoher Systemlast wird mit großer Wahrscheinlichkeit ein zur Abgabe von Last geeigneter Knoten gefunden.

Prozessorvergabe - vert. Syst.: Lastverteilungsalgorithmen

- **Symmetrisch initiiert:**
  - **Transport-Strategie:**
    - Schwellwert-Strategie, die auf der Länge der Bereitwarteschlange basiert
  - **Auswahl-Strategie: Irgendeine**
  - **Informations-Strategie:**
    - Informationsaustausch nur bei Bedarf
    - Mittlere Systemlast wird lokal bestimmt
    - Akzeptierbarer Bereich bestimmt das Stabilisationsvermögen des Algorithmus

Prozessorvergabe - vert. Syst.: Lastverteilungsalgorithmen

- **Plazierungs-Strategie:**
  - **Sender-initiiert**
    - Sender sendet “zu hoch”-Nachricht und wartet auf “empfangsbereit”
    - Empfänger sendet “empfangsbereit” und korrigiert seine Last mit der zu erwartenden
    - Nach Empfang von “empfangsbereit”: Falls immer noch Sender, verlagere günstigeren Prozeß
    - Falls kein “empfangsbereit”, dann per Broadcast “Mittelwerte ändern”.
  - **Empfänger-initiiert:**
    - Ein Empfänger versendet per Broadcast “zu wenig Last” und Wartet auf “zu viel Last”
    - Nach Empfang von “zu wenig Last” wird “empfangsbereit” versandt und die Last des Empfängers entsprechend erhöht.
    - Falls keine “zu wenig Last”-Nachricht empfangen, dann per Broadcast “Mittelwerte anpassen” um die Schätzung der mittleren Last anderer Prozessoren zu erniedrigen

Prozessorvergabe - vert. Syst.: Beispiele

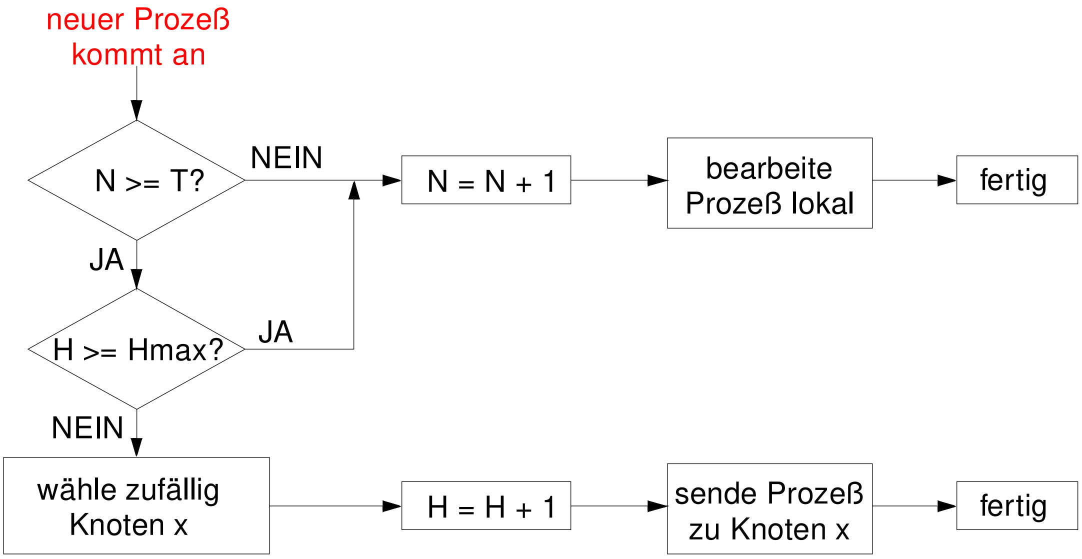
6.4 Beispiele

❑ **Algorithmus 1**

◆ **Strategien**

- **Transport-Strategie:**
  - Last-Schwellwert mit Transportschwellwert
- **Auswahl-Strategie:**
  - Irgendeine
- **Plazierungs-Strategie:**
  - Zufällig
- **Informations-Strategie:**
  - Informationsaustausch ausgelöst durch Sender

Prozessorvergabe - vert. Syst.: Beispiele



N: Länge der lokalen Bereitwarteschlange

T: Schwellwert für die lokale Bereitwarteschlange

H: Verlagerungszähler des Prozesses

Hmax: maximal erlaubte Verlagerungszahl

Prozessorvergabe - vert. Syst.: Beispiele

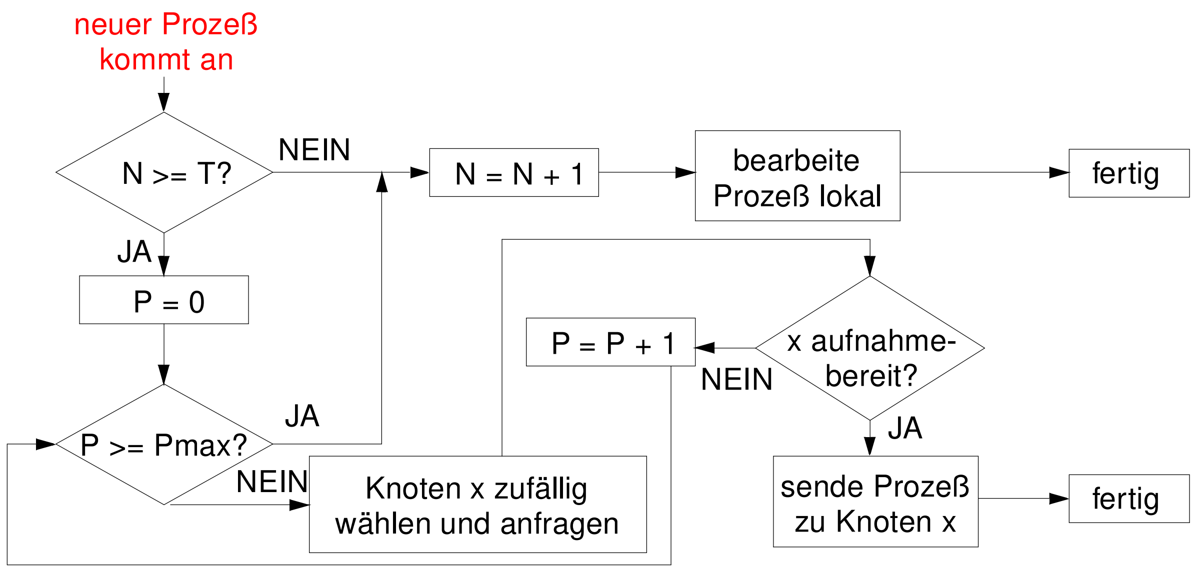
❑ **Algorithmus 2**

◆ **Strategien**

- **Transport-Strategie:**
  - Last-Schwellwert
- **Auswahl-Strategie:**
  - Irgendeine
- **Plazierungs-Strategie:**
  - Schwellwertbegrenzte zufällige Prüfung
- **Informations-Strategie:**
  - Informationsaustausch ausgelöst durch Sender

BP 2

Prozessorvergabe - vert. Syst.: Beispiele



N: Länge der lokalen Bereitwarteschlange

T: Schwellwert für die lokale Bereitwarteschlange

P: Versuchszähler

Pmax: maximal erlaubte Versuchszahl

BP 2

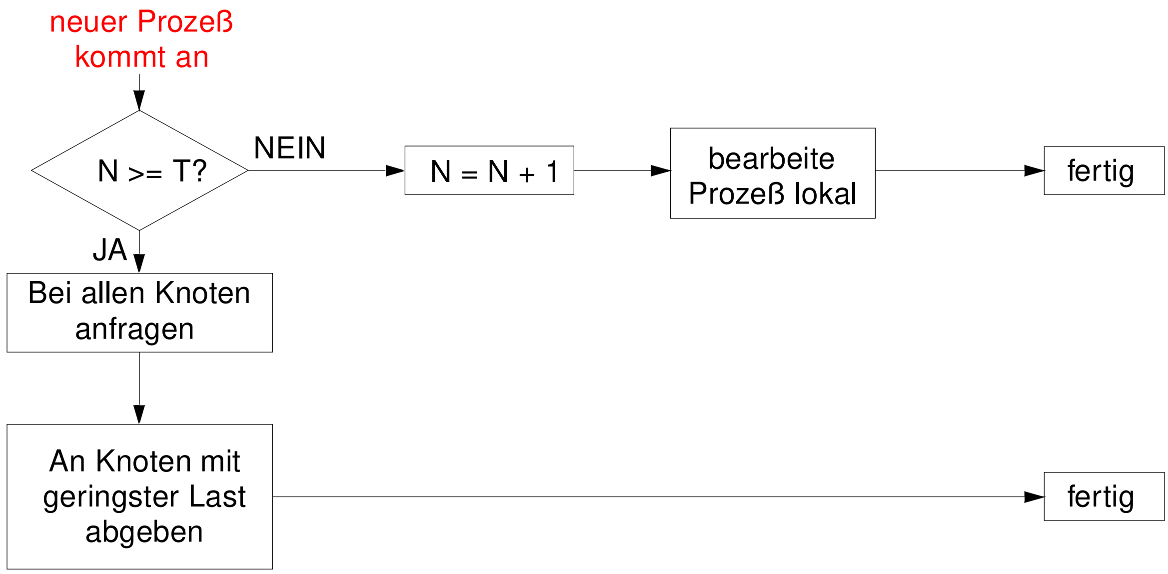
Prozessorvergabe - vert. Syst.: Beispiele

❑ **Algorithmus 3**

◆ **Strategien**

- **Transport-Strategie:**
  - Last-Schwellwert
- **Auswahl-Strategie:**
  - Irgendeine
- **Plazierungs-Strategie:**
  - Geringste Last
- **Informations-Strategie:**
  - Informationsaustausch ausgelöst durch Sender

BP 2 Prozessorvergabe - vert. Syst.: Beispiele



N: Länge der lokalen Bereitwarteschlange

T: Schwellwert für die lokale Bereitwarteschlange

BP 2 Prozessorvergabe - vert. Syst.: Beispiele

- **Hybrides Modell**
  - **Up-Down-Algorithmus**

Mutka, M. W.; Livny, M.: Scheduling Remote Processing Capacity In A Workstation-Processor Bank Network. Proceedings of the 7th International Conference on Distributed Computing Systems, Berlin, West Germany, September 21-25, 1987, pp. 2 - 9.

- **Jeder Arbeitsplatzrechner w besitzt einen Zuordnungsindex U(w) in einer zentralen Nutzungs-Tabelle**
- **Wenn der Besitzer von w unbearbeitete Aufträge für einen Poolrechner anstehen hat, bekommt U(w) negative, mit der Zeit zunehmende Strafpunkte**
- **Wenn der Besitzer von w Aufträge auf einem Poolrechner laufen hat, bekommt U(w) positive, mit der Zeit zunehmende Strafpunkte**
- **Wenn Rechner w keine Aufträge für den Pool anstehen hat und kein Poolrechner von ihm genutzt wird, wird U(w) näher an null gebracht bis es den Wert null hat.**
- **Heuristik: Ein freier Poolprozessor wird dem Auftrag zugeordnet, der den kleinsten Wert U(w) hat.**

Prozessorvergabe - vert. Syst.: Abhängige Aufträge

6.5 **Zuordnung bei nebenläufigen abhängigen Aufträge**

❑ **Fragenstellung**

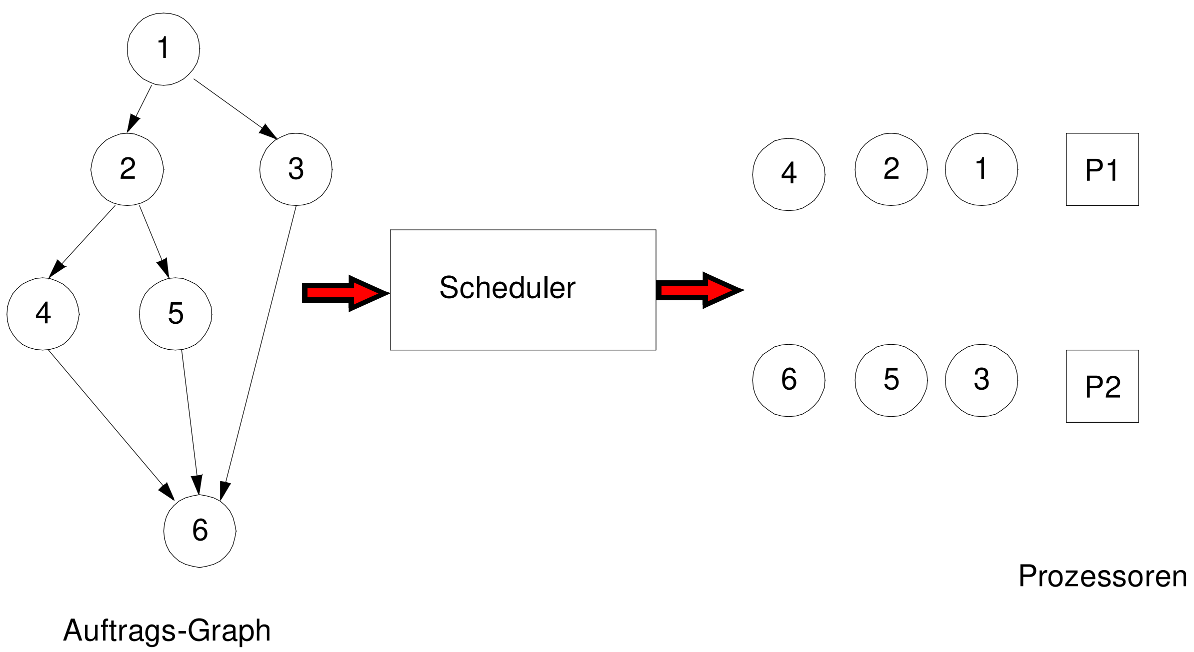
◆ **Gegeben ist**

- **eine Menge von Aufträgen mit Reihenfolgebedingungen und Forderungen an Rechenzeit und Kommunikatio**
- **eine Menge von Prozessoren, die über ein Kommunikationsnetzwerk verbunden sind**

◆ **Gesucht ist**

- **eine Zuordnung der Aufträge zu Prozessoren so, daß die Ausführungszeit minimiert wird.**

Prozessorvergabe - vert. Syst.: Abhängige Aufträge



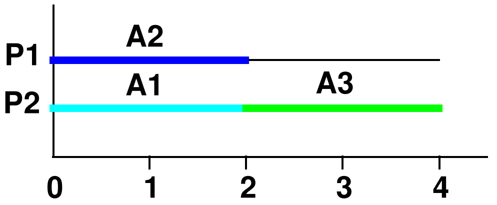
Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann
Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

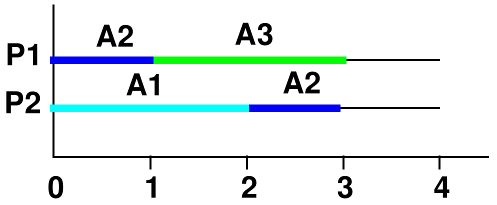
BP 2 Prozessorvergabe - vert. Syst.: Abhängige Aufträge

- Problem ist NP-vollständi
- In sehr speziellen Fällen gibt es zeitlich polynomial begrenzte Lösungsalgorithmen
  - Baumartiger Graph, wobei alle Aufgaben die gleiche Ausführungszeit haben
  - Im Fall von 2 Prozessoren
- Im allgemeinen müssen heuristische Verfahren benutzt werden

BP 2 Prozessorvergabe - vert. Syst.: Abhängige Aufträge

- Eigenschaften von Zuordnungsalgorithme
- Bewertungsmaßstäbe
  - Verweildauer der Gesamtaufgabe
  - Zusatzbelastung durch den Zuordnungsalgorithmus
- Einzelne Anwendung versus mehrere Anwendungen im Zeitmultiplex
  - Einzel-Anwendungen: Minimierung der Verweilzeit
  - Mehrere Anwendungen: Lastausgleich
- Verdrängend oder nicht-verdrängend





- Adaptiv oder nicht-adaptiv
  - Adaptive Zuordnung kann Laufzeitmessungen nutzen
  - Sammeln und Auswerten von Laufzeitinformationen erzeugt Zusatzaufwand

Prozessorvergabe - vert. Syst.: Abhängige Aufträge

❑ Clustering

◆ Aufgabenstellung

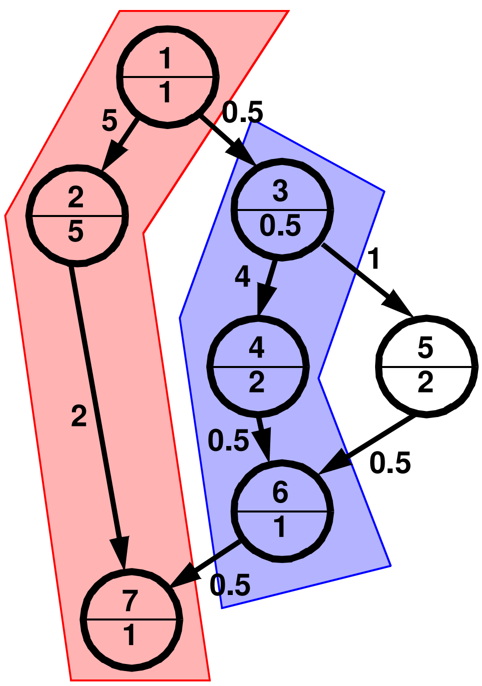
- Wenn zwei Aufträge dem gleichen Knoten zugeordnet werden, ist der Kommunikationsaufwand zwischen ihnen null.
- Zuordnung zu unterschiedlichen Knoten erhöht die Parallelität und erniedrigt die Verweildauer.

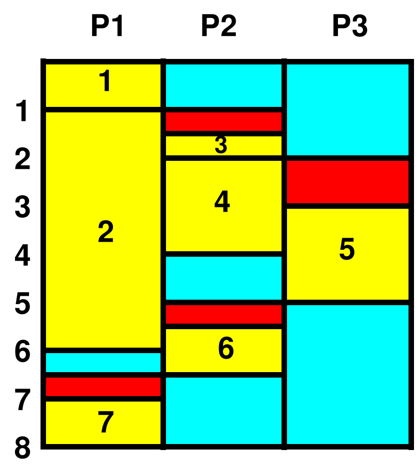
◆ Cluster-Algorithmen

- Aufträge werden in Cluster eingeteilt.
- Alle Aufträge eines Clusters werden dem gleichen Knoten zugeteilt
- Optimales Clustering ist NP-vollständi

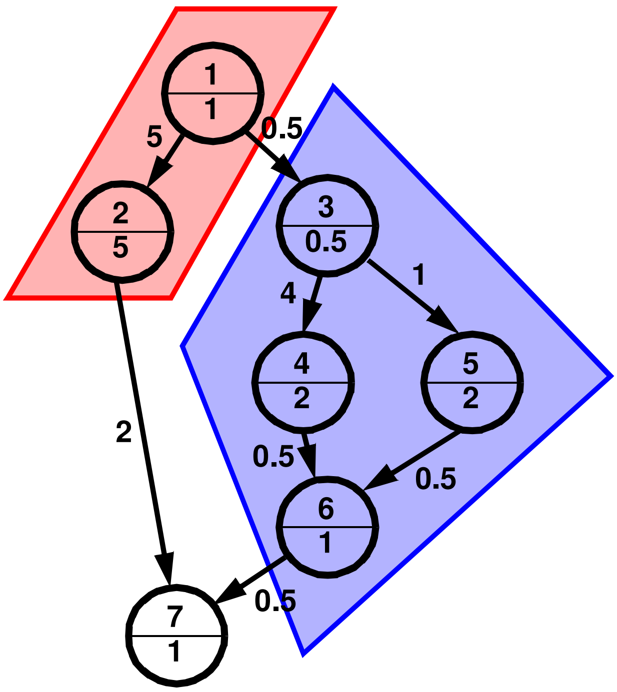
Prozessorvergabe - vert. Syst.: Abhängige Aufträge

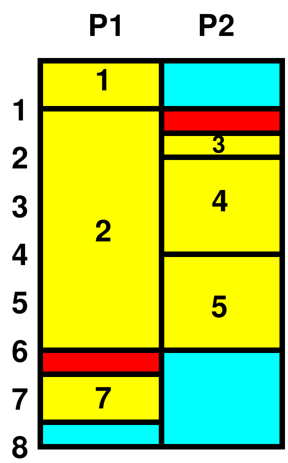
❑ Beispiele





Prozessorvergabe - vert. Syst.: Abhängige Aufträge





Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann