

Universität Karlsruhe (TH)

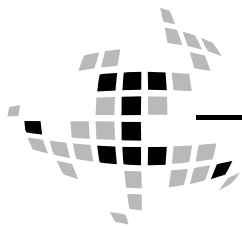
Forschungsuniversität · gegründet 1825

Betriebssysteme

Prof. Dr. Walter F. Tichy

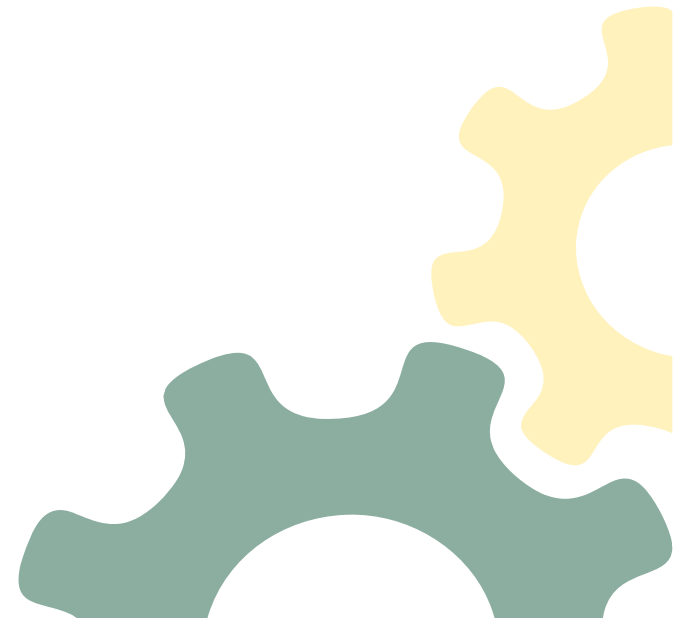
Dr. Victor Pankratius

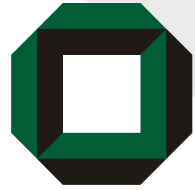
Ali Jannesari



Fakultät für **Informatik**

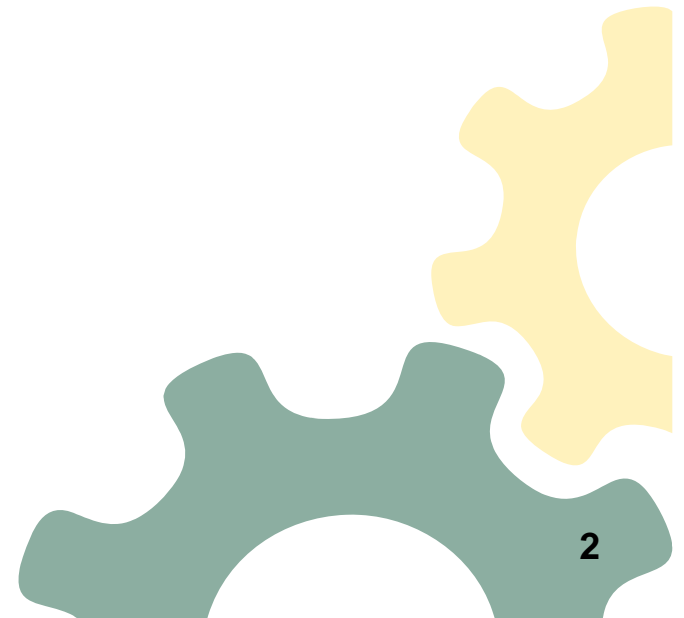
Lehrstuhl für Programmiersysteme

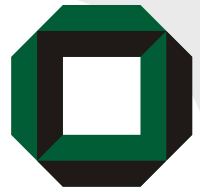




Vorlesung "Cluster Computing"

- Architektur von Rechnerbündeln
- **Betrieb von Rechnerbündeln**
 - Gesamtsicht (Single System Image)
 - Ablaufplanung (Scheduling)



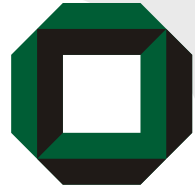


Gesamtsicht Single System Image – SSI

Benutzer wollen Rechnerbündel als Einheit nutzen

- Aufträge einmal starten, statt sich auf jedem Knoten (womöglich mit unterschiedlichem Passwort) anzumelden und jeweils einen oder mehrere Prozesse zu starten. Ebenso für's Beenden.
- Einheitliche Leistungsbeobachtung/-überwachung
- Einheitliche Ressourcen-Abrechnung
- Einheitliche Verwendung der E/A-Systeme (Dateisystem, ...)

- Abstraktion von Knotenanzahl und Leistungsstärke
- Automatische Zuteilung der Prozesse zu Prozessoren
- Automatische Lastbalance bei mehreren Nutzern



Anforderungsliste

- Ressourcenüberwachung

Verteiltes Programm mit Rechten, um auf interne Datenstrukturen aller Betriebssysteminstanzen zuzugreifen.

- Auftrags-Steuerung

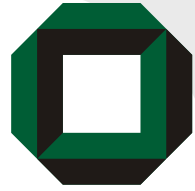
Erweiterung der lokalen Kennung zu einer globalen. Systemprogramme zum Starten (clusterrun) und zur Signalisierung (clusterkill). Dazu ist ein Dämon pro Knoten nötig.

- Globale Informationsbasis

Zusammentragen lokaler Status-Informationen

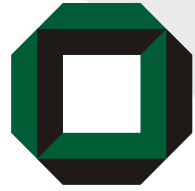
- Skalierbarkeit

Zentrale Steuerung skaliert schlecht. Verteilte Auslegung nötig. Verteile Steuerung erfordert mehr Kommunikation.



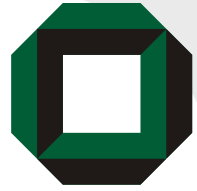
Verfügbare Systeme

- Stapelverarbeitung:
 - Condor, Wisconsin State University
 - PBS Portable Batch System, NASA Ames & LLNL
 - CCS Computing Center Software, PC², Paderborn
- Betriebssystemerweiterungen
 - **MOSIX Multicomputer OS for Unix**
- Verteilte Betriebssysteme
 - Chorus, Locus, Amoeba, Mach, Plan-9, ...
- Parallele Programmierumgebungen



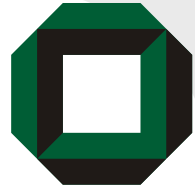
MOSIX - Multicomputer OS for Unix

- **Ziel:**
Cluster Computing soll so einfach und effizient wie die Verwendung von SMP Maschinen werden.
- **Vorgehen:**
Kernerweiterung (kernel patch), die die im Bündel vorhandenen Ressourcen verwaltet und dem Benutzer die Illusion eines Einzelsystems vorspiegelt:
 - transparent für Applikationen
 - keine spezielle Bibliotheken
 - Entfernte Ressourcen können wie lokale Ressourcen angesprochen und verwendet werden.
- <http://www.mosix.org>



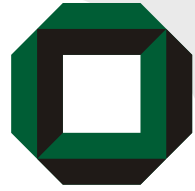
Zentrales MOSIX-Konzept: Prozessmigration

- **Lastbalancierung**
Verwendung leer laufender CPUs zur Leistungssteigerung
- **Migration von langsamen zu schnelleren Prozessoren, falls verfügbar**
- **Verkürzung der Speicherlatenz:**
Wenn die lokal laufenden Prozesse zu viel Hauptspeicher benötigen, ist Prozessmigration auf anderen Knoten ggf. eine schnellere Alternative zu Seitenwechsel (-flattern)
- **Schnelle Dateioperationen:**
Prozess kann schneller auf Datei zugreifen, wenn er auf dem Rechner läuft, auf dessen lokaler Platte die Datei liegt.
- **Schnellere Interprozesskommunikation:**
Auf einem Rechner laufende Prozesse können über gemeinsamen Speicher schneller kommunizieren als über das Netz.
- **Netzhauptspeicher und DSM:**
Wenn Daten speicherübergreifend auf verschiedenen Knoten liegen, sollten Prozesse zu ihren Daten wandern (Lokalität).



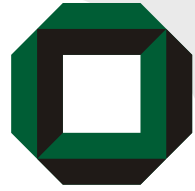
MOSIX: Basistechnologie

- **Preemptive Prozessmigration:**
Jeder Prozess kann zu jeder Zeit im System vollkommen transparent migriert werden.
- Migration wird durch **adaptive Algorithmen** ausgelöst, die bei Erreichen von **Schwellwerten** tätig werden.
- Skalierbarkeit durch **probabilistische Verfahren** unter Verwendung von partiellem Wissen.
- Daher ist MOSIX gut geeignet für Anwendungen mit unvorhersehbaren Laufzeiten und Ressourcenanforderungen.



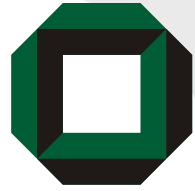
MOSIX: Probabilistische Verwaltung

- Jeder Knoten schickt **periodisch** seine Lastinformation an eine zufällig ausgewählte Teilmenge aller Knoten.
- Alle Entscheidungen werden von den Knoten lokal getroffen. Sie basieren auf einem Fenster der zuletzt von anderen Knoten empfangen Lastinformationen.
- Ansatz ist **robust** und **skalierbar**, da er unabhängig von der Anzahl an Knoten nur von der Größe der bei einer Aktion berücksichtigten Knotenteilmenge abhängt.



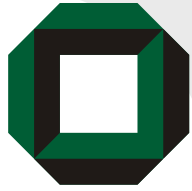
MOSIX: Lastbalancierung

- Wenn der Last-Schwellwert erreicht ist, dann wird Last aller Knoten im Beobachtungsfenster gesichtet. Minimierung der paarweisen Unterschiede durch Abgeben von Prozessen.
 - anpassbar auf CPU und Netzwerkleistung (wird beim Hochfahren gemessen)
 - adaptive Ressourcenzuteilung besser als jedes statische Verfahren
 - fast optimale Ressourcenausnutzung



MOSIX: Internes Verfahren

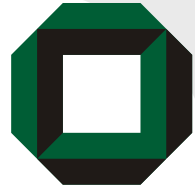
- Prozess wird in zwei Teile zerlegt:
 - Prozesskontext (Code, Keller, Daten)
 - Systemkontext (Systemumgebung)
- Nur der Prozesskontext wird migriert.
- Der Systemkontext bleibt auf dem initialen Knoten (*Heimat-Knoten*), der sämtliche ortsgebundenen Systemaufrufe ausführt und so die ursprüngliche Systemumgebung sicherstellt
 - Systemaufrufe werden zu entfernten Methodenaufrufen
 - Dateizugriffe, Sockets, ... bleiben im Heimatknoten.
- Zusätzlich wird ein Kommunikationskanal geöffnet für asynchrone Ereignisse (z.B. Signale).



Mosix: Internes Verfahren (2)



- **Ausnahmen:**
 - Ortsungebundene Systemaufrufe werden lokal ausgeführt.
 - Ein/Ausgabekanäle werden verlegt, um Nähe zu Dateien auszunutzen.



Vorlesung "Cluster Computing"

- Architektur von Rechnerbündeln
- Betrieb von Rechnerbündeln
 - Single System Image
 - Ablaufplanung (Scheduling)

