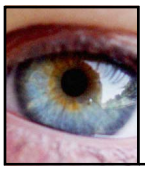
Dynamische Skalierbarkeit

Alexander Eichhorn
Verteilte Systeme und Betriebssysteme
Technische Universität Ilmenau

Frühjahrstreffen der GI Fachgruppe Betriebssysteme
30. Juni 2005
Koblenz



Vortragsüberblick

Teil 1	Teil 2	Teil 3
		
Begriffs-bestimmungen	Muster skalierbarer Systeme	Skalierbarkeit Virtueller Maschinen

Skalierbarkeit

Qualitative Aussage

- Fähigkeit zur Anpassung an wechselnde Lasten ohne Änderung der zugrunde liegenden Algorithmen und Architekturen

Quantitative Aussage

- Verhältnis aus Wachstum der Leistungsfähigkeit zu Wachstum des Ressourcenbedarfs

- Gegenstand der Skalierung
  - Anzahl an Komponenten eines Systems (Software oder Hardware)
  - Durchsatzrate (Anzahl und Größe abgeschlossener Aufträge pro Zeit)
  - Antwortzeit (mittlere Bearbeitungsdauer eines Auftrags)
  - Datenvolumen
  - Fehlerraten

Bewertung der Skalierbarkeit

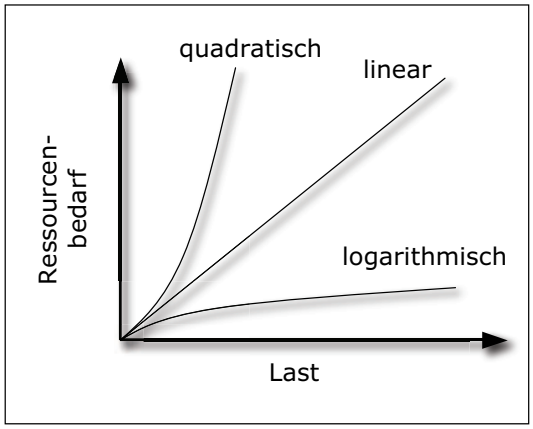
Sagt man: „Ein System *skaliert*

- konstant,
- linear,
- logarithmisch,
- quadratisch oder
- exponentiell

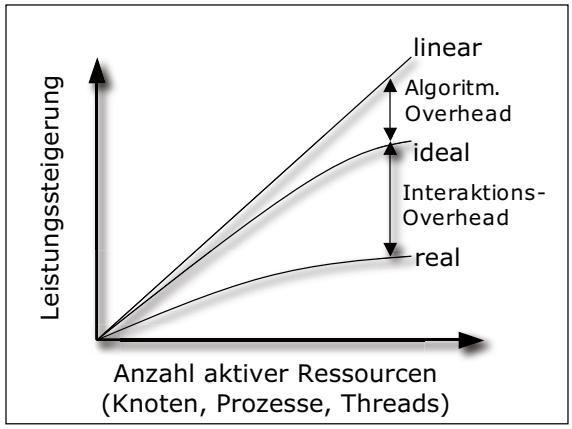
[mit wachsender Last]“,
so meint man damit *den Zuwachs des Ressourcenverbrauchs*.

Bewertung der Skalierbarkeit

Abhängigkeit von Last und Ressourcenbedarf

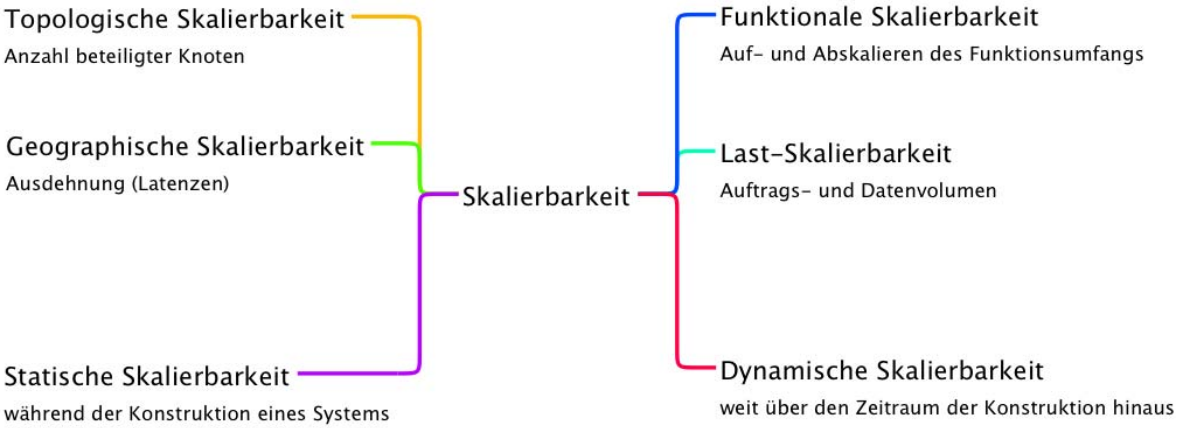


Erzielbare Leistungssteigerungen



Interaktions-Overhead:
- Latenzen
- Synchronisation
- Scheduling
- Cache-Effekte

Formen der Skalierbarkeit



Teil 1: Begriffe und Zusammenhänge

Skalierbarkeit von Algorithmen

O-Notation

- zur Beurteilung der
  - Rechenzeitkomplexität
  - Speicherplatzkomplexität
  - Botschaftenkomplexität (bei verteilten Algorithmen)

- z.B. in der C++ Standard Template Library (STL) angegeben
  - `std::list<>::insert()` = O(1)
  - `std::map<>::operator[]` = O(log n)

Teil 2: Muster skalierbarer Systeme

Problemwelt

Welche Probleme sind besonders gut skalierbar?

- unabängige Webserver-Anfragen
- Datenspeicherung (RAID, Parallele Dateisysteme)
- Multimediadatenströme
- Matrizenmultiplikation

Welche Probleme sind nicht skalierbar?

- Probleme mit hohem sequenziellen Anteil
  - Prozesssteuerungen
  - ?

Lösungswelt

Prinzipen zum Bau skalierbarer Systeme

- Nebenläufigkeit und Asynchronität
- Zustandslosigkeit
- Dezentrale Strukturen
- Caching und Replikation
- Gesunder Menschenverstand

Nebenläufigkeit und Asynchronität

- Blinder Einsatz von Threads bringt nicht automatisch Skalierbarkeit
- Vermeiden von Wartesituationen (Asynchrone IO, IO-Batching)
- Subtile Zusammenhänge im Detail
- Entwickler werden von heutigen Threadkonzepten damit alleine gelassen
- Ein Großteil der grundlegenden Techniken bleibt Handarbeit

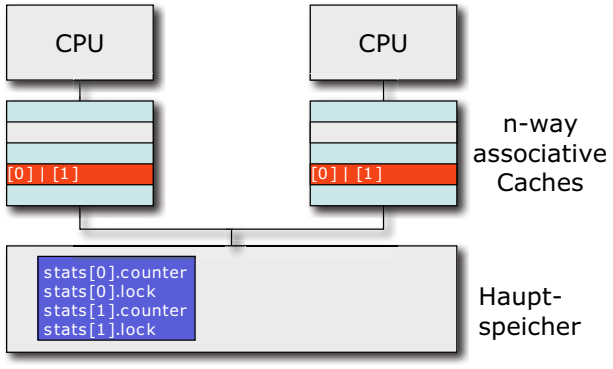
Nebenläufigkeit

- Nötige Handarbeit, um Threadkonzepte skalierbar einzusetzen
  - Aufteilung großer Aufgaben in kleine Teilaufgaben
  - Reduktion gemeinsamer exklusiver Ressourcen zu Gunsten privater
  - Feingranulare Locks, Read-Write Locks
  - Vermeiden von False-Sharing

```
struct statistics_data {
   int counter;
   pthread_mutex_t lock;
};

statistics_data stats[NUM_CPUS];
```

Beispiel: zu enges Layout von privaten Daten oder Locks bei SMP und NUMA



Teil 2: Muster skalierbarer Systeme

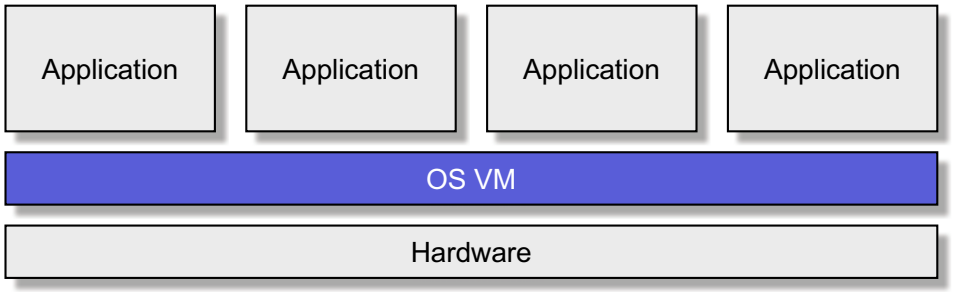
Weitere Prinzipien

Prinzipen zum Bau skalierbarer Systeme

- Zustandslosigkeit
  - Vermeiden von Speicherbedarf und Zustandsabfragen
  - Bsp: Internet, select() vs. poll()
- Caching
  - Vermeiden von Kommunikation zu Lasten von Konsistenz
- Replikation
  - Verminderung von Anfragelast durch organisierte Verteilung der Last
- Dezentrale Strukturen
  - Vermeiden von Flaschenhälsen durch organisierte Verteilung der Last
  - Bsp: Peer2Peer, Parallele Dateisysteme, Hierarchien (DNS)
- Gesunder Menschenverstand
  - jeder Algorithmus und jede Architektur haben eine Achillesverse (beachten und Einsatz bewusst abwägen)
  - Einsatz von Profilern (Cache Misses, Pipeline-Stalls, TLB-Misses)

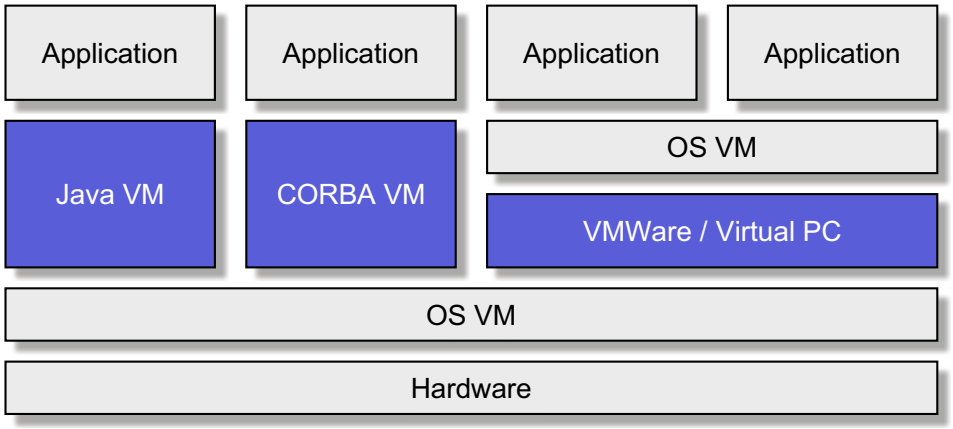
Teil 3: Virtuelle Maschinen

Die Steinzeit



- Abstraktion der Hardware durch ein Betriebssystem
  - Prozesse, Threads
  - Adressräume
  - IPC und IO
  - Dateien

Gestern



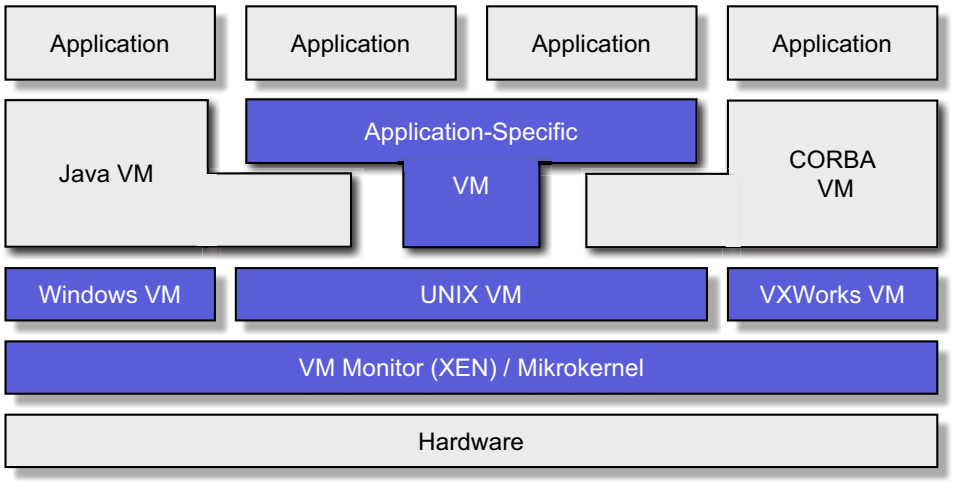
Plus: Applikationsnahe Dienste

- RPC und RMI
- Verteilte Dienste, Garbage Collection, Objektmobilität

Plus: Virtualisierte Hardware

- Emulierte CPU mit MMU
- Emulierte und reale Dateisysteme, IO-Hardware

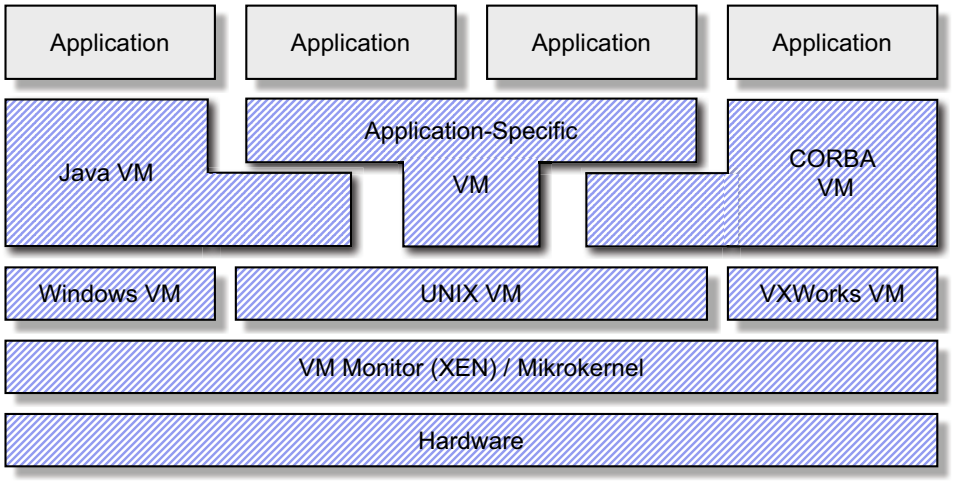
Heute und Morgen



Plus:

- Applikationsspezifische Virtuelle Maschinen (funktionale Skalierbarkeit)
- Stapelbare VM: freie Wahl der BS-Persönlichkeit und Eigenschaften (Sicherheit, Echtzeit, Multimedia,...)
- Verschiebbare VM - Skalierung physischer Ressourcen bei Bedarf

Und Übermorgen?



Plus:

- Neue Hardware: Multicore-CPU's (CELL), IO-MMU's
- Schichtübergreifende Koordination von Strategien zwischen gestapelten VM's

Skalierbarkeit

Performanzprobleme

- Wieviele VM's verträgt eine Hardwareplattform?
- Wie hoch dürfen die Stapel werden?

Organisatorische Probleme

- Wie muss die schichtübergreifende Kommunikation von
  - Informationen über hardwarenahe Belange
  - Strategien des Ressourcenmanagements

  geschehen?
- Brauchen wir dafür neue Abstraktionen, Schnittstellen und Primitive zwischen VM's und zu den Applikationen?

Zusammenfassung
Resultate

Skalierbarkeit

★ *Verschiedene Formen der Skalierbarkeit unterscheiden*

★ *Skalierbarkeit muss auf und über alle Ebenen hinweg explizit durch konstruktive Maßnahmen hergestellt werden*

Offene Fragen

- Sind unsere aktuellen Thread-Abstraktionen den künftigen Umgebungen gewachsen?
  - Skalierbare und ereignisbasierte Systeme
  - Multicore-Prozessoren
- Brauchen wir neue Abstraktionen, Schnittstellen und Primitive zwischen Systemebenen (VM's) und Applikationen?
  - Koordination von (Ressourcenmanagement-)Strategien
  - Informationen über systemnahe Details

Vielen Dank für Ihre
Aufmerksamkeit

Sie sind nun herzlich zur Diskussionsrunde eingeladen.