Empirische Ermittlung Cache-bedingter Umschaltverluste

Robert Kaiser

Labor für Verteilte Systeme
Fachhochschule Wiesbaden

GI/ITG Frühjahrstreffen, Wiesbaden, 11.03.2008

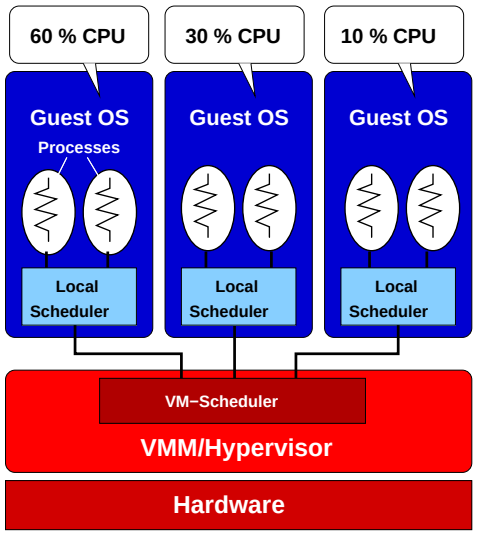
Inhalt

Inhalt

Motivation

- Ziel: Virtualisierung von Echtzeitsystemen
- Virtual Machine Monitor (VMM) agiert (u.A.) auch als Scheduler
- i.d.R. anteilige Zuweisung der CPU-Leistung (*proportional share*)
- ideal: Pro VM kontinuierlich verfügbare CPU-Anteile
- real: Approximation der Kontinuität durch „schnelles" Umschalten
- Frage: wie schnell?

$\Rightarrow$ Kompromiss: Kontinuität $\leftrightarrow$ Umschaltkosten



Ansatz

- Umschaltkosten können –wenn bekannt– in Echtzeitplanung berücksichtigt werden:
  - in Simulation
  - in on-line-Scheduler
  - auch von Interesse für Mehrprozessor-Systeme

⇒ Wünschenswert: Umschaltkosten beziffern, bzw. formal beschreiben

- aber: Analytische Ermittlung nur in Ausnahmefällen praktikabel:
  - Vollständige Kenntnis über Zustand und Verhalten der VMs erforderlich
  - Berechnung für on-line-Scheduler zu komplex
- daher: Umschaltkosten werden häufig vernachlässigt
  (dabei keine Klarheit über Gültigkeitsgrenzen dieser Annahme!)

⇒ Versuch: Empirische Herangehensweise:

- Beobachten/Messen des Systemverhaltens
- Approximative Beschreibung des Verhaltens durch Funktionen

Inhalt

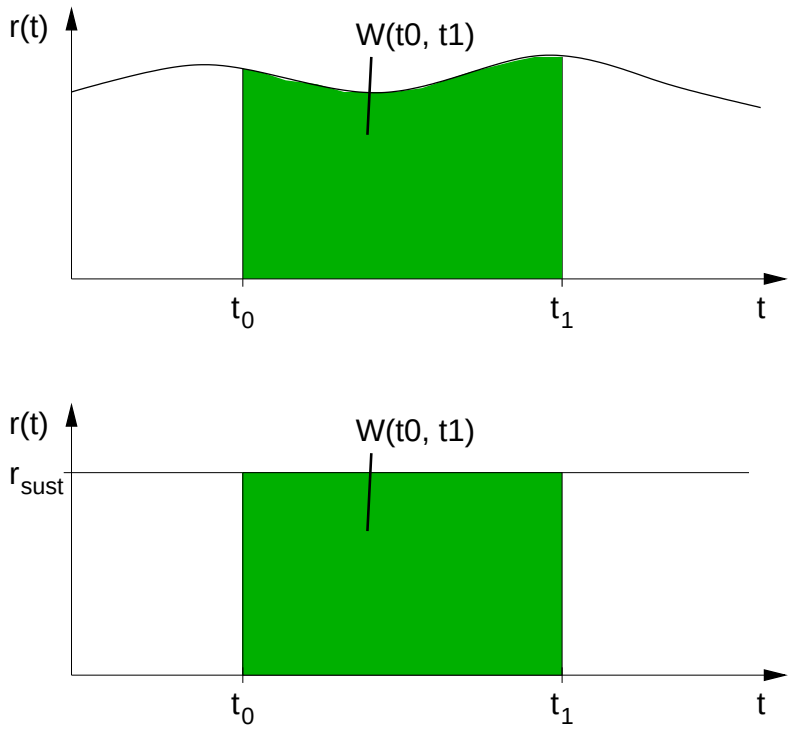
Begriffe

- Rechenleistung, bzw. Fortschrittsrate: $r(t)$
- Arbeit: $W(t)$

$$W(t_0, t_1) = \int_{t_0}^{t_1} r(\tau) d\tau$$

- Un-unterbrochene Ausführung einer konstanten Rechenlast (d.h. $r(t) = r_{sust}$):

$$W(t_0, t_1) = (t_1 - t_0) \cdot r_{sust}$$



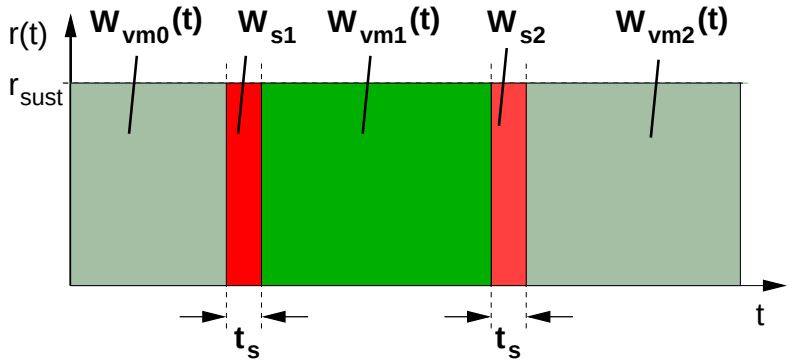
Verluste durch Virtualisierung

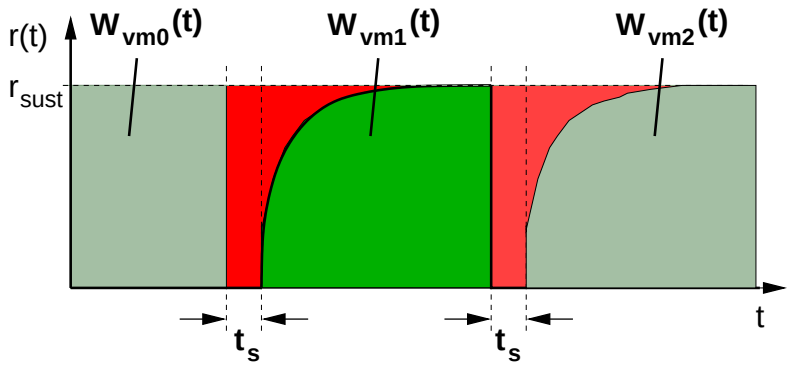
- Bei Ausführung in einer VM entstehen „Ausfallzeiten” durch
  - Aktivität anderer VMs (kein Verlust)
  - Scheduler-Aktivität (Verlust)
- Annahme: Scheduler-Laufzeit konstant (= t_s)

$\Rightarrow$ Kosten pro Scheduler-Aufruf:

$$W_s = t_s \cdot r_{sust}$$

- Bei Prozesswechsel[a]: weitere Kosten
- Beide können VM zugeordnet werden





[a] (N.B.: Prozesswechsel $\neq$ Scheduler-Aufruf)

Inhalt

Prozesswechselkosten

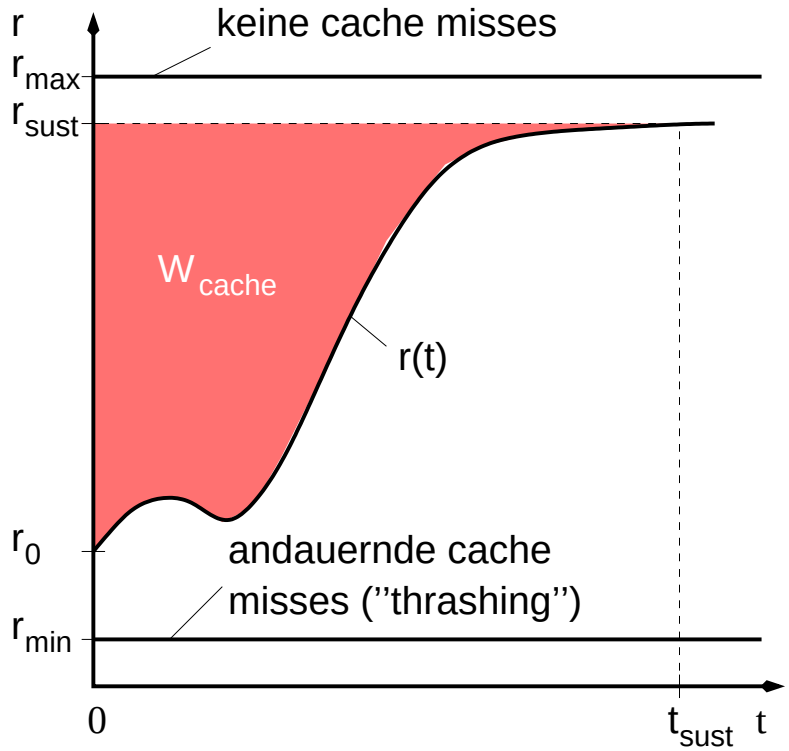
- Prozesswechselkosten: Bedingt durch Cache-/TLB-misses
- Kein diskretes Zeitfenster, sondern kontinuierliche „Verlangsamung" der CPU, d.h. verminderte Fortschrittsrate

$\Rightarrow$ Kosten pro Prozesswechsel:

$$W_{cache}(t) = t \cdot r_{sust} - \int_0^t r(\tau) d\tau$$

- Relativer Verlust:

$$O_{cache}(t) = 1 - \frac{1}{t} \int_0^t \frac{r(\tau)}{r_{sust}} d\tau$$



Prozesswechsel bei $t = 0$

Nutzlastanteil

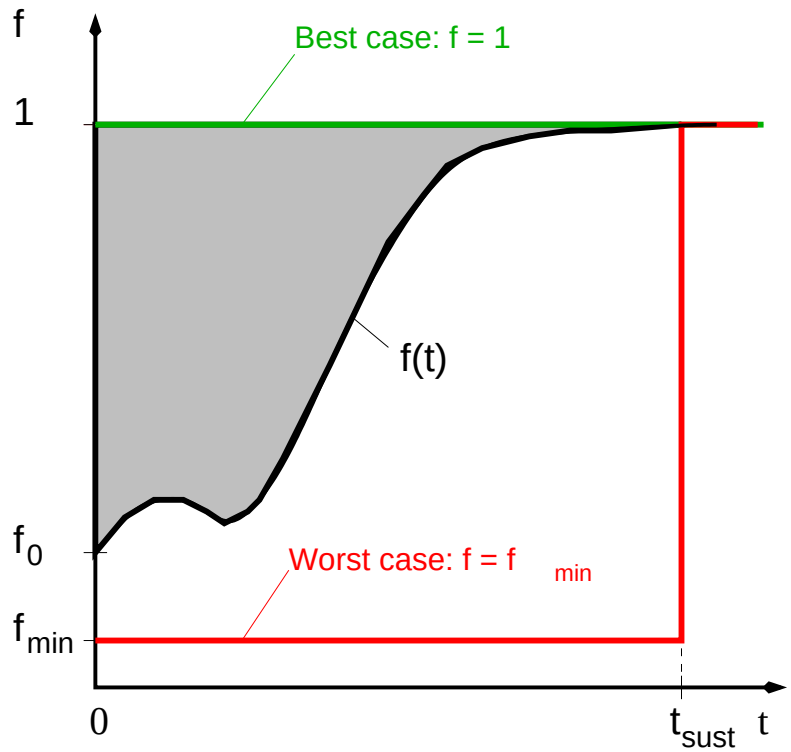
- Verhältnis von genutzter zu gesamter Rechenleistung:

$$f(t) := \frac{r(t)}{r_{sust}}$$

- Damit:

$$O_{cache}(t) = 1 - \frac{1}{t} \int_0^t f(\tau) d\tau$$

- Problem: $f(t)$ kann nicht allgemein angegeben werden, aber
  - best case: $f(t) = 1$
  - worst case: $f(t) = f_{min} > 0$
  - real: dazwischen..

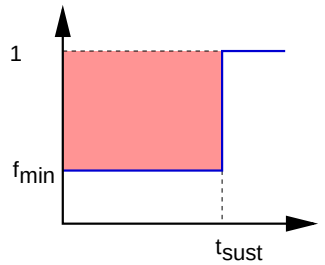


Prozesswechsel bei $t = 0$

Approximation des Nutzlastanteils

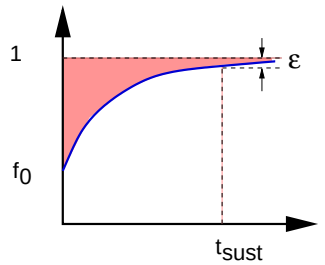
- Annahme eines zeitlichen Verlaufes des Nutzlastanteils
- Beschreibung durch (integrierbare) Zeitfunktion $f(t)$, z.B.:
  - „cache flooding” (worst case) ...

$$f_{flood}(t) = \begin{cases} f_{min}, & 0 \leq t < t_{sust} \\ 1, & t \geq t_{sust} \end{cases}$$



  - ... oder Exponentialfunktion ...:

$$f_{avg}(t) = 1 + (f_0 - 1) \cdot e^{-kt}$$



  - ....

$\Rightarrow$ Berechnung der Verluste pro Umschaltvorgang möglich

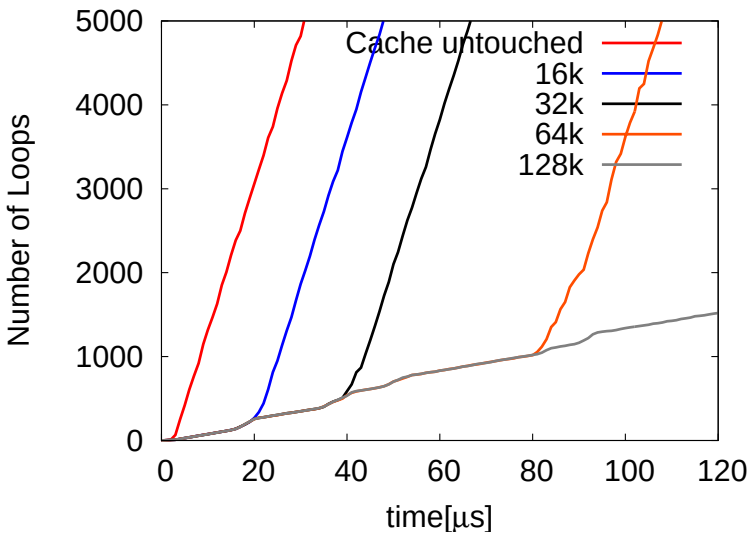
- Wahl der Funktion $f(t)$ und ihrer Parameter individuell für jede VM, z.B.
  - „harte” Echtzeit $\longrightarrow$ wähle $f_{flood}(t)$
  - „weiche” Echtzeit $\longrightarrow$ wähle $f_{avg}(t)$

Inhalt

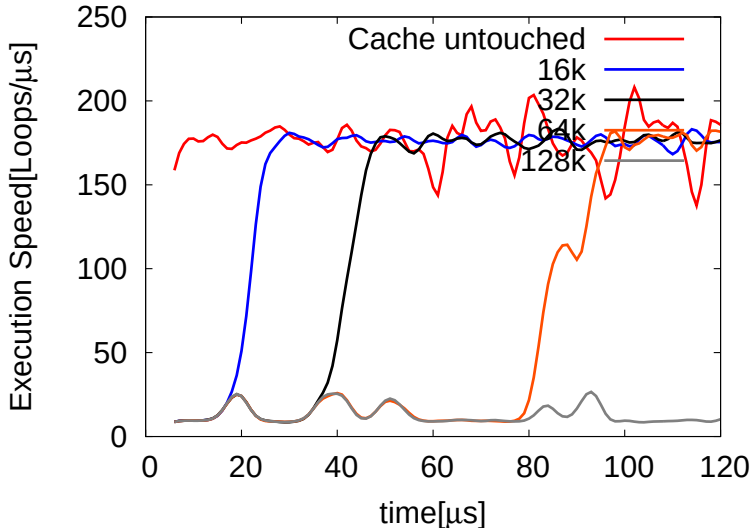
Ziele und Messverfahren

- Frage: Wie sind die Parameter (t_{sust}, f_{min}, etc.) zu wählen?
- Praktischer Ansatz: Messen, dabei 2 Ziele:
  1. Erproben/Validieren des Worst Case („flooding”)
  2. Ermitteln realistischer Parameter
- Vorgehensweise:
  - Caches in definierten Zustand bringen (löschen, schreibend/lesend füllen)
  - Schreib- bzw. Lesezugriffe auf einen zuvor nicht im Cache befindlichen Datenbereich konfigurierbarer Größe („working space”)
  - Messen: Zeit für eine vorgegebene, variable Anzahl Zugriffe
- Testumgebung:
  - Verschiedene IA-32 Maschinen: Celeron@2.5 GHz, PentiumM@1.5GHz
  - Linux (Ubuntu, Kernel 2.6.20)
  - Messung mit gesperrten Interrupts (Testcode in Gerätetreiber)

Beispiel: Celeron



- Zunächst gemessen: Arbeit



- Abgeleitet: Fortschrittsrate (= Rechenleistung)

Ergebnisse

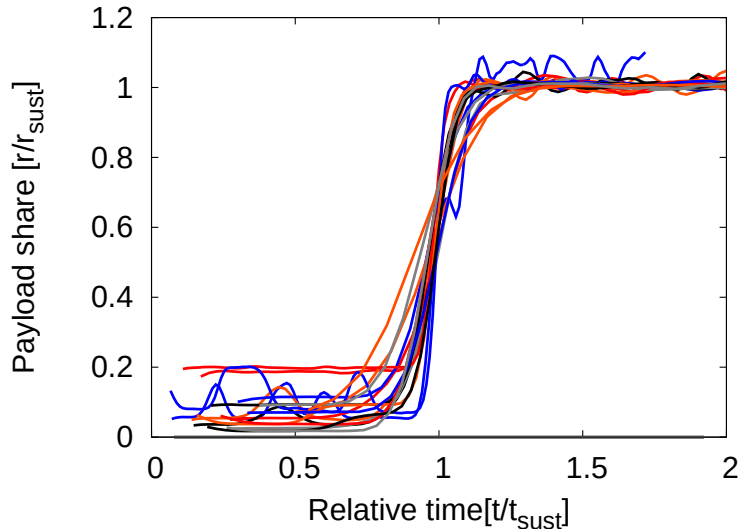
- Messergebnisse:

Maschine	WSS	Cache	f_0	t_{sust}
Cel.@2.5G	16k	dirty	0.11	$22\mu s$
Cel.@2.5G	32k	dirty	0.11	$43\mu s$
Cel.@2.5G	64k	dirty	0.09	$85\mu s$
Cel.@2.5G	16k	invd	0.17	$12\mu s$
Cel.@2.5G	32k	invd	0.18	$20\mu s$
Cel.@2.5G	64k	invd	0.20	$35\mu s$
PentM@1.5G	16k	dirty	0.09	$25\mu s$
PentM@1.5G	32k	dirty	0.08	$38\mu s$
PentM@1.5G	64k	dirty	0.12	$91\mu s$
PentM@1.5G	16k	invd	0.17	$16\mu s$
PentM@1.5G	32k	invd	0.12	$29\mu s$
PentM@1.5G	64k	invd	0.21	$55\mu s$

Testfall: Schreibzugriffe auf aufeinanderfolgende Cache-Zeilen
invd = Cache zuvor invalidiert, dirty = Cache zuvor gefüllt

- t_{sust} wächst mit „working space”
  (aber nicht proportional ..)
- f_0 zwischen (hier) 8% und 21%
  (kein klares Bild ..)

- Normierte Darstellung:

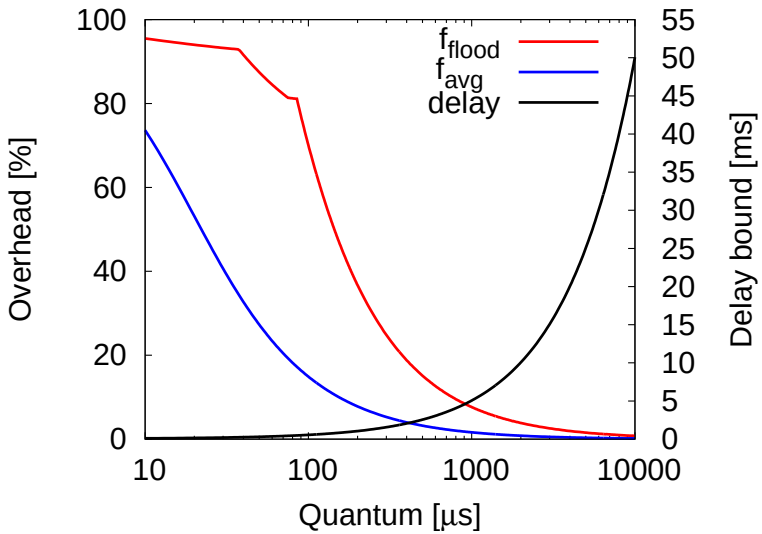


⇒ Entspricht qualitativ den Erwartungen

⇒ Zahlenmäßige Ergebnisse erfordern weitere Untersuchung

Anwendungsbeispiel: Simulation eines VTRR Schedulers

- VTRR: O(1) Scheduler für anteilige Zuweisung[a]



(3 VMs, CPU: Celeron@2.5GHz, 64k WSS)

- „Delay bound”: Maximale Verzögerung einer VM
- Proportional zur Zeitscheibengröße

⇒ Kontinuität ↔ Umschaltkosten jetzt zahlenmäßig erfasst

⇒ Für die gegebene Konfiguration (Celeron@2.5GHz) sind bei Zeitscheibengrößen unter ca. 1 ms die Verluste nicht mehr vernachlässigbar!

[a] Siehe: Nieh et al: *Virtual-Time Round-Robin: An O(1) Proportional Share Scheduler*

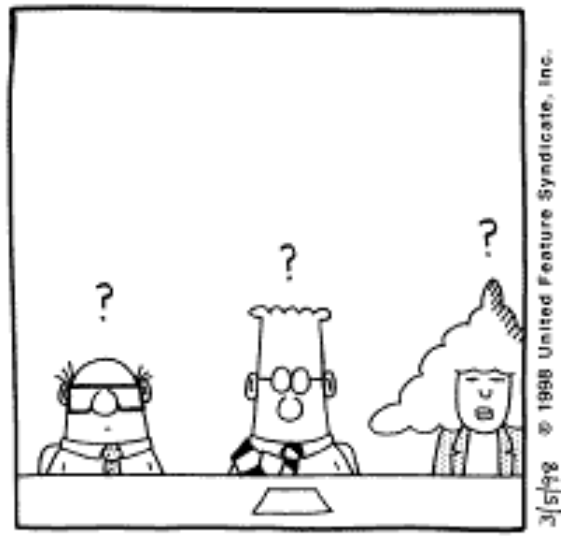
Inhalt

Zusammenfassung/Ausblick

- Ergebnisse:
  - Modell zur approximativen Berechnung von Umschaltverlusten
  - Liefert individuelle Umschaltverluste für jede VM
  - Approximation wahlweise (pro VM) pessimistisch/optimistisch
  - Pessimistische Schätzung praktisch reproduziert
- Weitere Arbeiten
  - Überprüfung der Messungen auf Architektur mit nur einem Cache Level
  - Entwicklung und Validierung von Approximationsfunktionen für „durchschnittliche” Szenarien
  - Anwendung innerhalb einer Virtualisierungsumgebung

Das Ende

Danke für die Aufmerksamkeit!



... Fragen?