technische universität
dortmund

DA-Abschlussvortrag

Anwendungsspezifische Maßschneiderung des Linux-Kerns für virtuelle Maschinen

Robert Neue

Betreuer: Jochen Streicher, Prof. Dr. Olaf Spinczyk
Arbeitsgruppe Eingebettete Systemsoftware

Lehrstuhl für Informatik 12
TU Dortmund

Robert.Neue@tu-dortmund.de

Agenda

- Einführung
- Grundlagen
- Systemanalyse
- Implementierung
- Evaluation
- Zusammenfassung und Ausblick

Einführung - Hintergrund

Virtuelle Maschinen

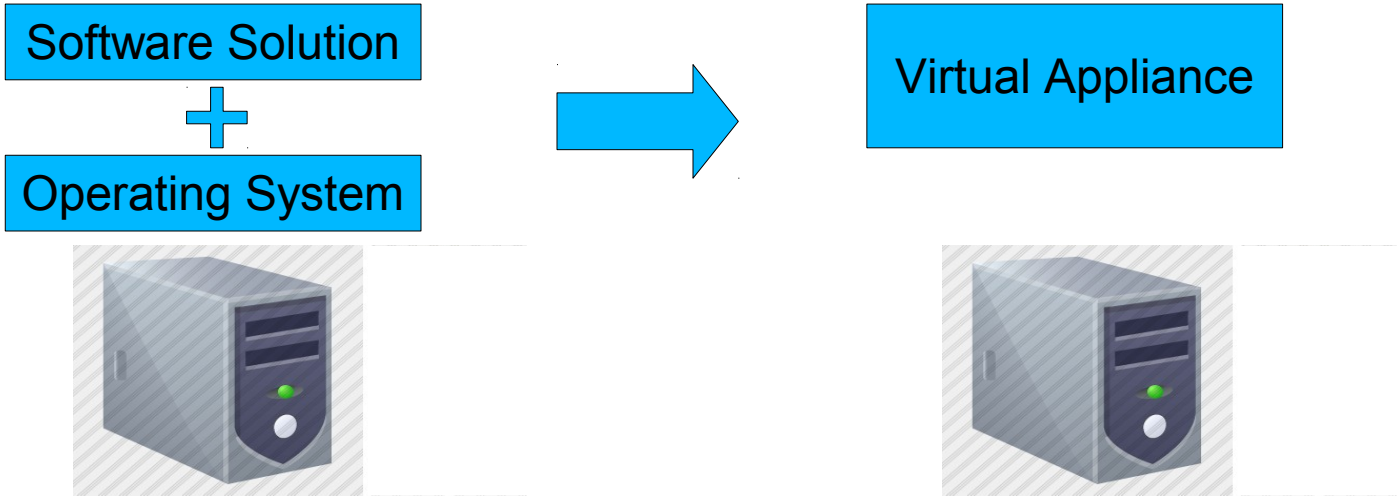
- Finden sich in vielen Bereichen (Server im Rechen-zentrum, Testplätze von Entwicklern, beim Planen von Migrationen)
- Virtuelle Maschinen versprechen eine bessere Auslastung vorhandener Hardware, Synergieeffekte durch Serverkonsolidierung
- Mehr und mehr Produktivrechner werden virtualisiert

(Selbst bei unternehmenskritischen Anwendungen wie etwa SAP-Server beträgt der Anteil der virtualisierten Server bereits 28%- 37% [Quelle: RAAD])

Einführung - Hintergrund

Ein Trend: Virtual Appliances

- General Purpose Operating Systems (GPOS) wie Linux, werden häufig virtualisiert



- Der Normalfall bei Serverkonsolidierung und Virtual Appliances ist ein kleines zuvor definiertes Set an Anwendungen

Einführung - Probleme

Betriebssysteme

- … sind optimiert auf auf einen abstrakten Normalfall – nicht auf eine einzelne Anwendung
- GPOS sind auf alle Eventualitäten vorbereitet
  - bösartige Programme
  - Mehrbenutzerbetrieb
  - Vielfädige Programme, miteinander interagierende Prozesse
- Alle Anwendungen zahlen dafür, indem sie mehr bekommen als sie benötigen (zusätzliche Checks, Indirektionen,etc.)

Einführung - Vision

- Einsatz von GPOS in bekannter Umgebung
  - Virtuelle Hardware,
  - Benutztes Dateisystem
  - Anwendung
- Anwendungsspezifische Anforderungen können durch Profiling ermittelt werden
- Einsparpotenziale durch die Maßschneiderung
  - Mehr-Prozessor-Unterstützung
  - Linux-Sicherheitskonzepte (DAC und LSM)
  - Zugriffe auf Dateisystem für vorhandenes optimieren
  - Entfernen nicht benötigter Subsysteme (Audio, USB, ...)

Einführung - Vision

- Anwendung profitiert stark von maßgeschneidertem Kernel
- Server die mehrere virtuelle Maschinen hosten, können
  - entweder kostengünstiger
  - oder leistungsfähiger gestaltet werden
- Leistungsfähigere Anwendungen könnten mit geringeren Taktraten auskommen und somit bei geringerem Stromverbrauch selbes leisten

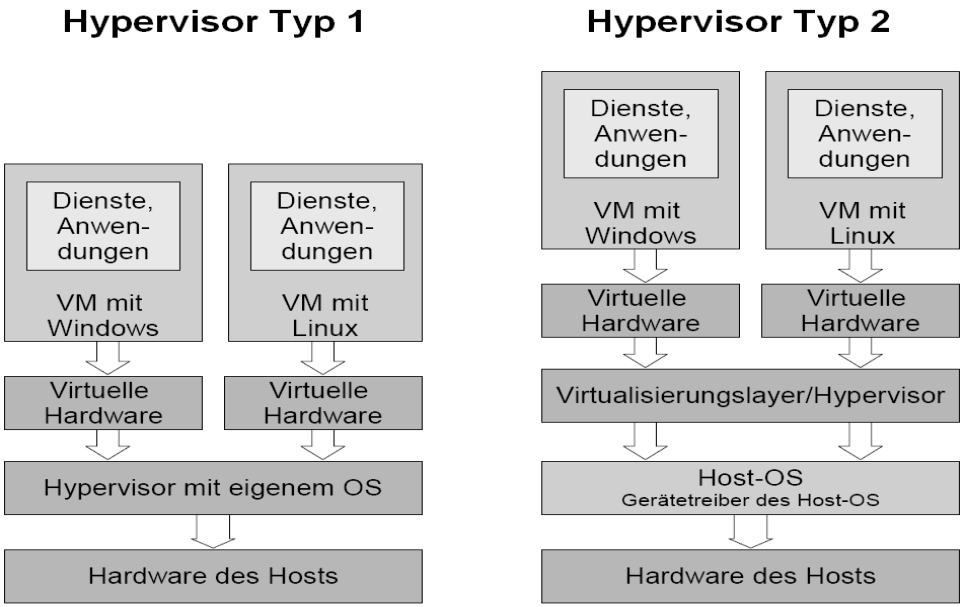
Agenda

- Einführung
- Grundlagen
- Systemanalyse
- Implementierung
- Ergebnisse
- Zusammenfassung und Ausblick

Grundlagen

Virtualisierungsarchitekturen

- Hypervisor Typ 1 (ohne Host OS): VMWare ESX,XEN
- Hypervisor Typ 2 (mit Host OS): VMWare Server, KVM



Grundlagen

Virtualisierungsarchitekturen

- Alleine der Overhead der Verwaltung des virtualisierten RAMs beträgt ca. 5-10% [Ahnert2009]
- Hinzu kommt die Emulation sensitiver Befehle durch den Virtualisierungslayer
- Einige Virtualisierungstechniken, wie etwa die Paravirtualisierung verändern sogar das Gast-Betriebs-system selbst

starker Einfluss der eingesetzten Virtualisierungslösung auf die Performance des Betriebssystems und der Anwendung

Grundlagen

Profiling:

- Kprobes
  - Debugging und Monitoring-Mechanismus zum Betreten und Verlassen beliebiger Punkte im Kontrollfluss
  - Geringer Overhead
- SystemTap
  - Skriptsprache für Kprobes
  - Unterstützung um schnell und mit wenig Overhead Daten sammeln zu können
  - Messbar sind Callgraphen, Dauer, Häufigkeit, Scheduler-Entscheidungen, etc.

Grundlagen

Betriebssystemoptimierungen - Kommerzielle Ansätze?

- Keine für automatische Kernelanpassungen
- Wenige auf Kernelkonfigurationsebene, meist nur für Spezialsysteme wie z.B. MeasureWare (HP-UX)
- Allgem. Lösung SARCheck
  - Optimierung von Kernelparametern (z.B. für DB´s)

Grundlagen – Kommerzielle Tools

SARCheck

- Application-based Profiling
- Vorschläge für verbesserte Parameterwerte
- Hilfestellung zur Änderung der Parameterwerte

Table of Contents

- Recommendations Section
- Resource Analysis Section
- Capacity Planning Section
- Custom Settings Section
- Summary of Statistics

SUMMARY

When the data was collected, no CPU bottleneck could be detected. A moderate memory bottleneck was seen.

RECOMMENDATIONS SECTION

All recommendations contained in this report are based solely on the conditions which were present when the perf
may cause some of these recommendations to result in worse performance. To minimize this risk, analyze data fror

Add memory. Additional memory may improve performance. If possible, borrow some memory for test purp
installation.

Change the bdflush parameter 'nfract' from 30 to 40. This is the percentage of dirty buffers allowed in the buffer cache before the kernel flushes some of them.

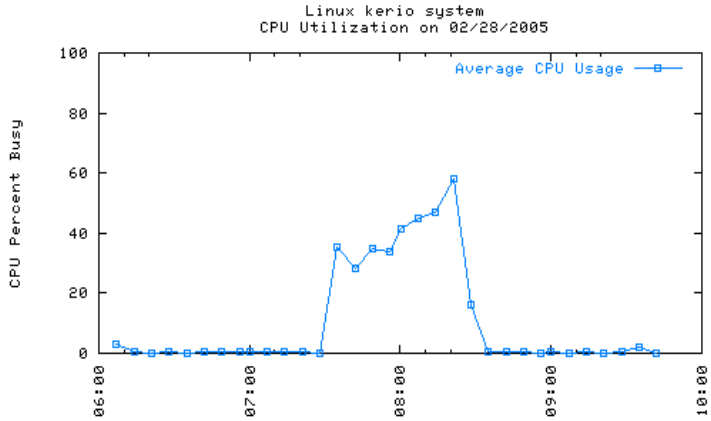
Change the bdflush parameter 'ndirty' from 64 to 128. This is the number of dirty blocks written to disk at one time when the bdflush daemon wakes up.

Change the bdflush parameter 'nrefill' from 64 to 512. This will allow the operating system to obtain more clean buffers when refill_freelist() is called.

Change the bdflush parameter 'nref_dirt' from 256 to 2048. This is recommended in order to keep its value 4 times higher than nrefill.

To change the value of the bdflush parameters immediately as described in the above recommendations, use the following command:

```
echo "40 128 512 2048 500 3000 60 0 0" > /proc/sys/vm/bdflush
```

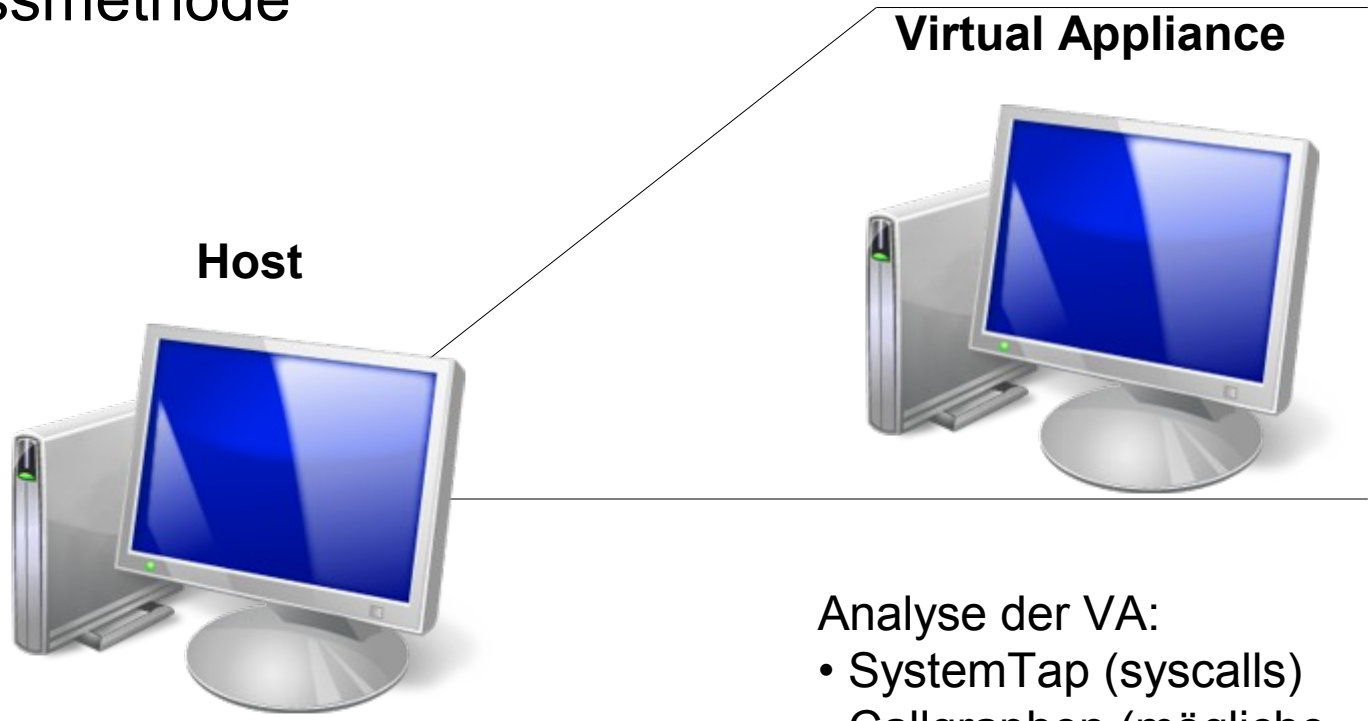


Agenda

- Einführung
- Grundlagen
- Systemanalyse
- Implementierung
- Evaluation
- Zusammenfassung und Ausblick

Systemanalyse

Messmethode



Kontrollieren der Gesamtperformance:
- Oprofile (Performance Counter)
- Autobench (HTTP Stresstest: Httperf)
- SystemTap (Scheduler-Entscheidungen für VMWare)

Analyse der VA:
- SystemTap (syscalls)
- Callgraphen (mögliche Pfade)
- Konstante Parameter
  - ↳ Bypass

Systemanalyse

Statische Analyse:

- Virtuelles File System
- Block I/O Scheduler
- Kernel Mode / User Mode
- Virtualisierung

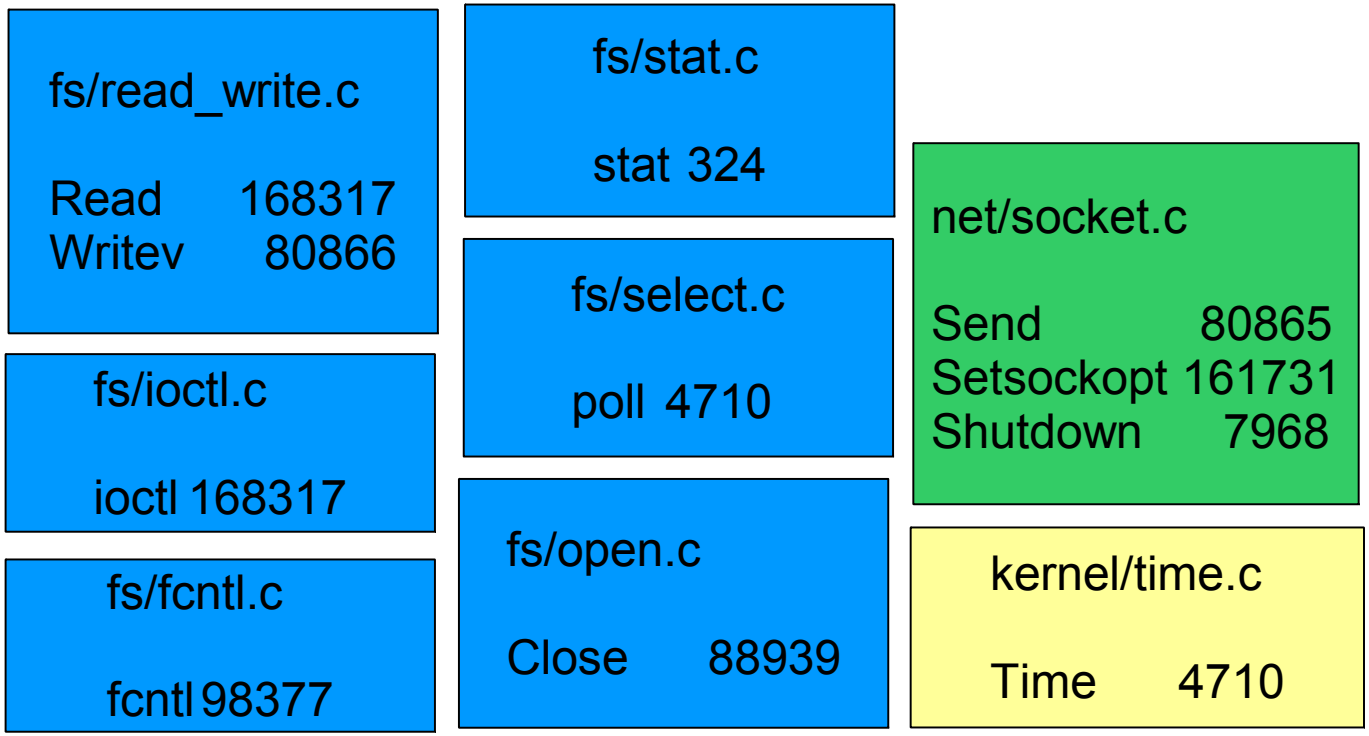
Dynamische Analyse:

- Benötigte Systemaufrufe
- Aufrufgraphen

Systemananlyse

HTTP-Server Lighttpd: Benötigt nur 11 syscalls

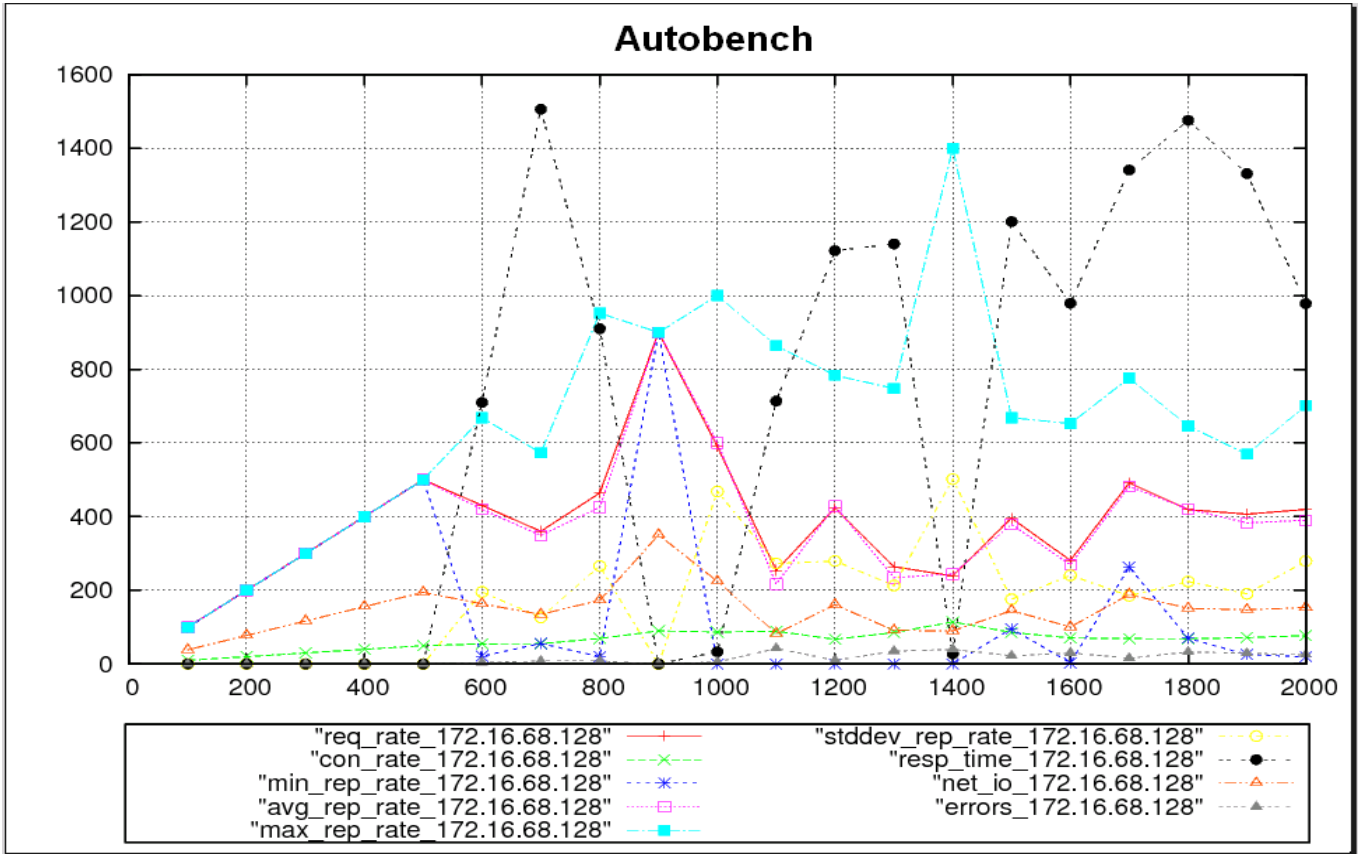
- Vor Deinstallation der restlichen Dienste: 61 Systemaufrufe



Systemanalyse – Beispiel lighttpd

Hostsystem

- Gesamtperformance der Anwendung überprüfen
  - Wie verhalten sich Antwortzeiten, etc. ?



Systemanalyse

Optimierungsmöglichkeiten:

Agenda

- Einführung
- Grundlagen
- Systemanalyse
- Implementierung
- Evaluation
- Zusammenfassung und Ausblick

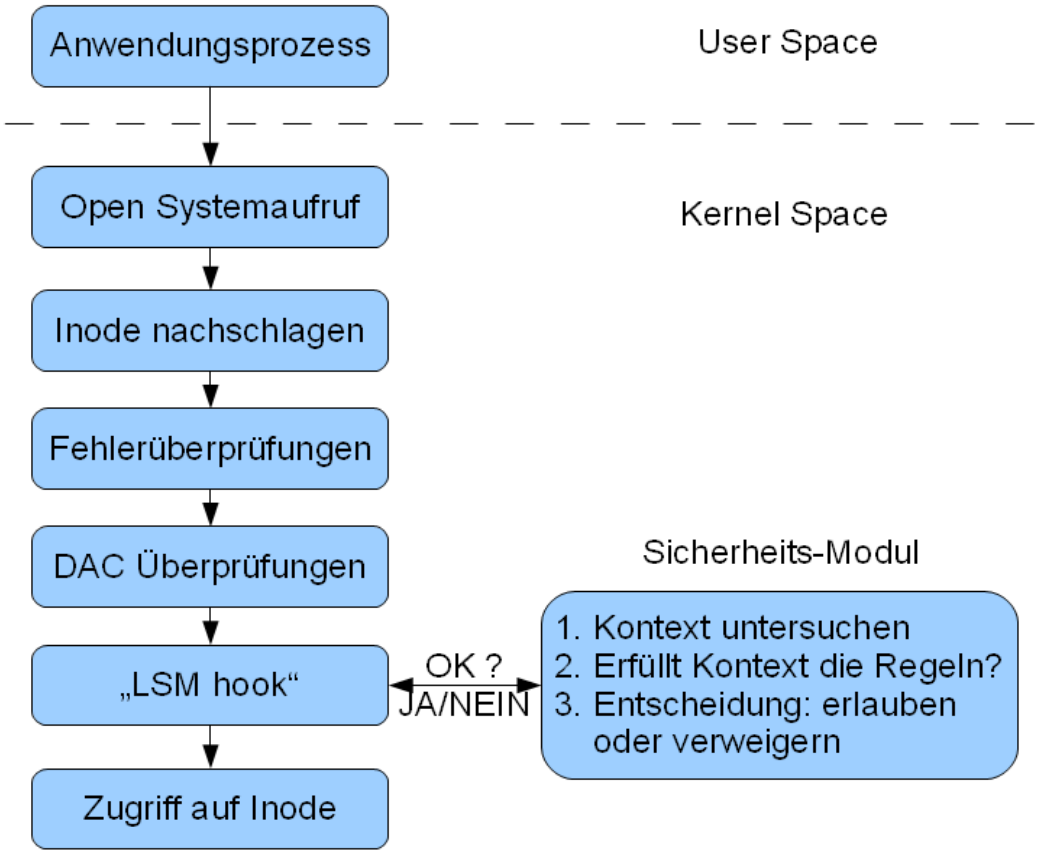
Implementierung

Maßschneiderung

- Bypass des Virtuellen Filesystem
- Entfernen der Verwaltung der Zugriffsrechte (DAC)
- Entfernen der Linux Security Modules
- Entfernen überflüssiger Checks
  - Abfrage auf spezielle Read-Only Dateisysteme in inode_permissions()
  - Abfrage, ob Dateisystem mit noexec geöffnet wurde

Implementierung

Unterschied DAC und LSM



Implementierung

Bordmittel

- GCC Compiler Optimierungen
  - -O2, -funroll-loops, -finline-functions
- Paravirtualisierung
  - VMWare: VMI-Schnittstelle nutzen

Implementierung

Kernel Mode / User Mode Grenze (Adressraumtrennung)

Idee: mit Kernel Mode Linux lighttpd direkt im Kernel ausführen.

- KML erlaubt Programme des User-Space im Kernel-Mode auszuführen.
  - Zahlreiche Synergieeffekte
- Ist jedoch in virtuellen Maschinen nicht anwendbar (CS-Register, Ring Deprivileging)

Agenda

- Einführung
- Grundlagen
- Systemanalyse
- Implementierung
- Evaluation
- Zusammenfassung und Ausblick

Evaluation

Testumgebung

- SystemTap
- Ksplice

Evaluation

Messgröße

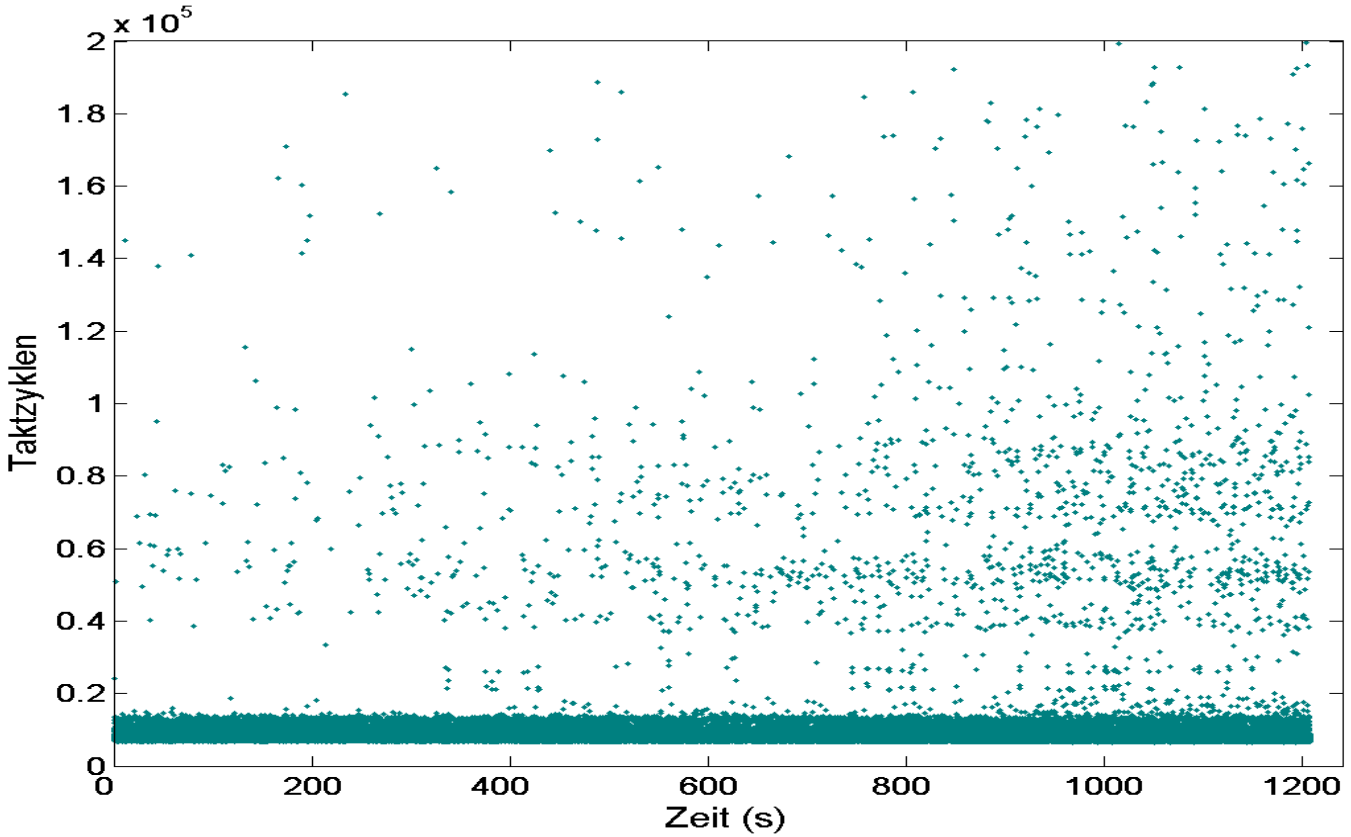
- Für maschinenunabhängige Aussagen: Taktzyklen
  - Wesentlich genauer (Zeit in µs – Takte ns bis ps)

Messung

- Interrupts können auftreten
- Gemessene Taktzyklen werden stark erhöht
  - Unterste (schnellste) 30 Taktzyklenzahlen werden gemittelt
  - Typische Maßzahl für Idealfall ohne Unterbrechungen

Evaluation

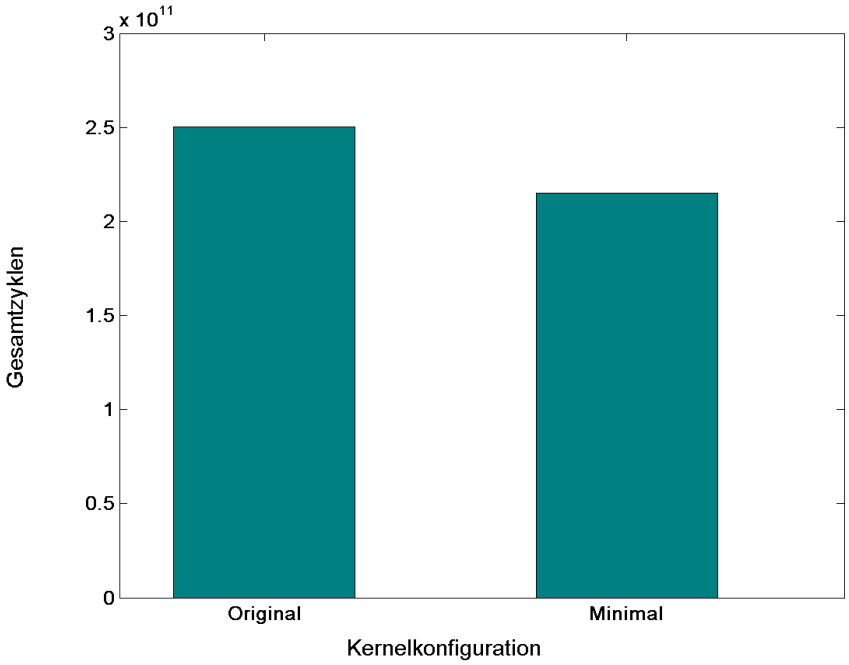
- Interrupts kaum ein Problem:



Evaluation

Auswirkungen der Bordmittel

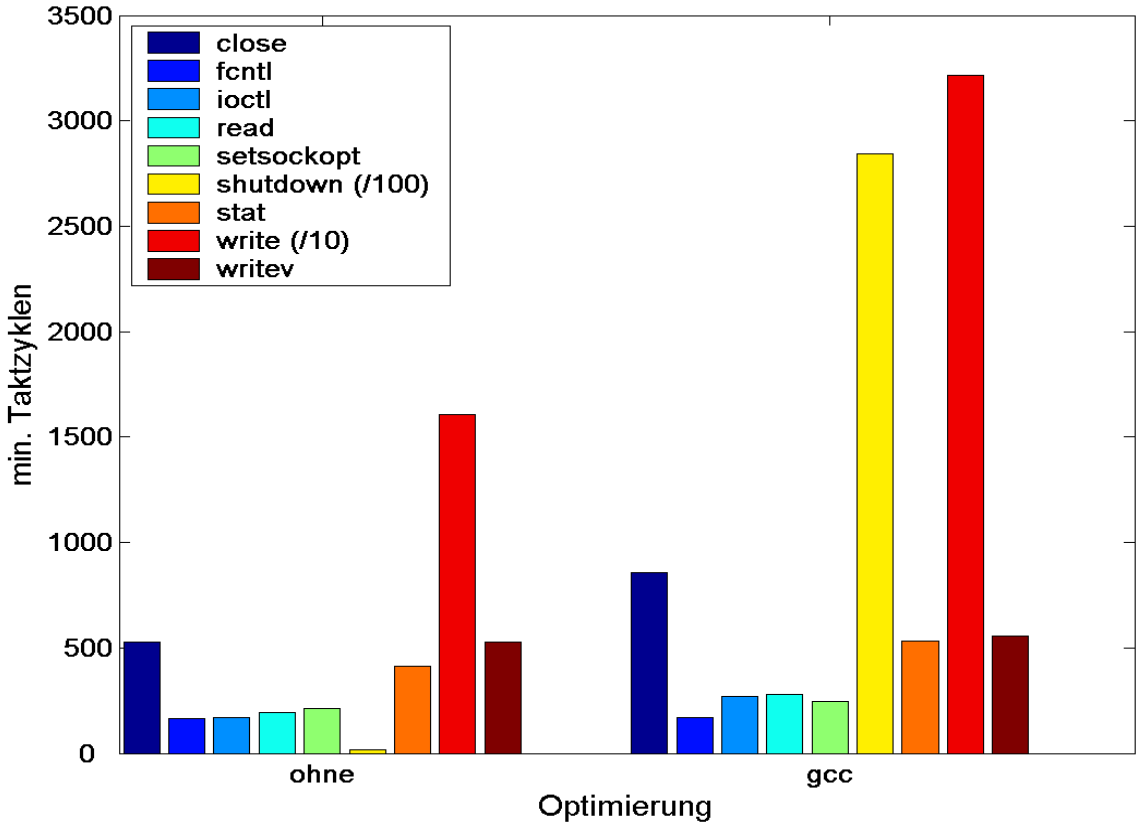
- Überflüssige Programme entfernen +0,04%
- Minimieren der Kernelkonfiguration +11%
  - Entfernen von Sub-systemen (Audio,etc.)
  - Module,
  - Multiprozessorunter-stützung
  - etc.



Evaluation

Auswirkungen der Bordmittel

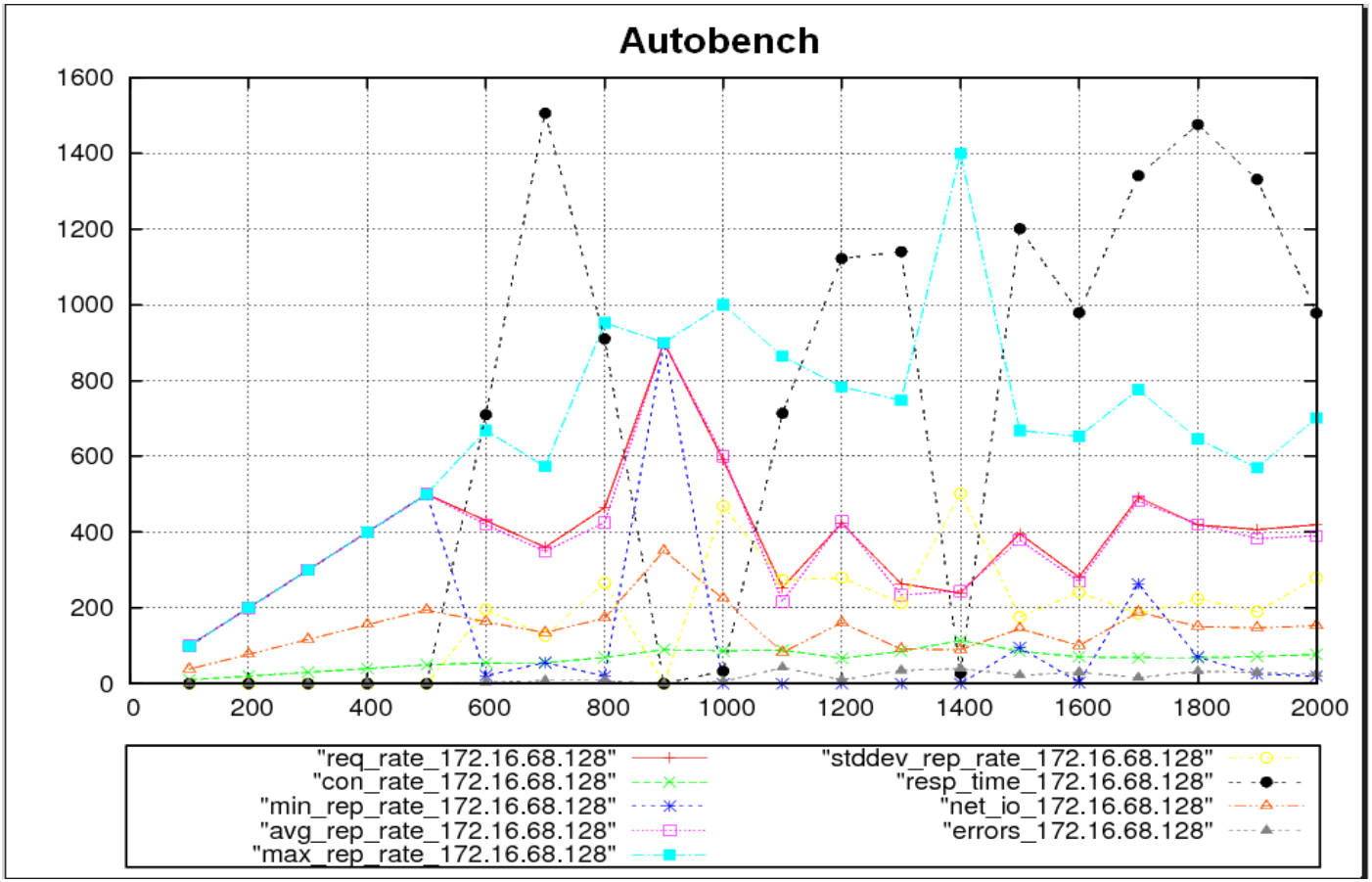
- GCC Optimierungen



Evaluation

Auswirkungen der Bordmittel

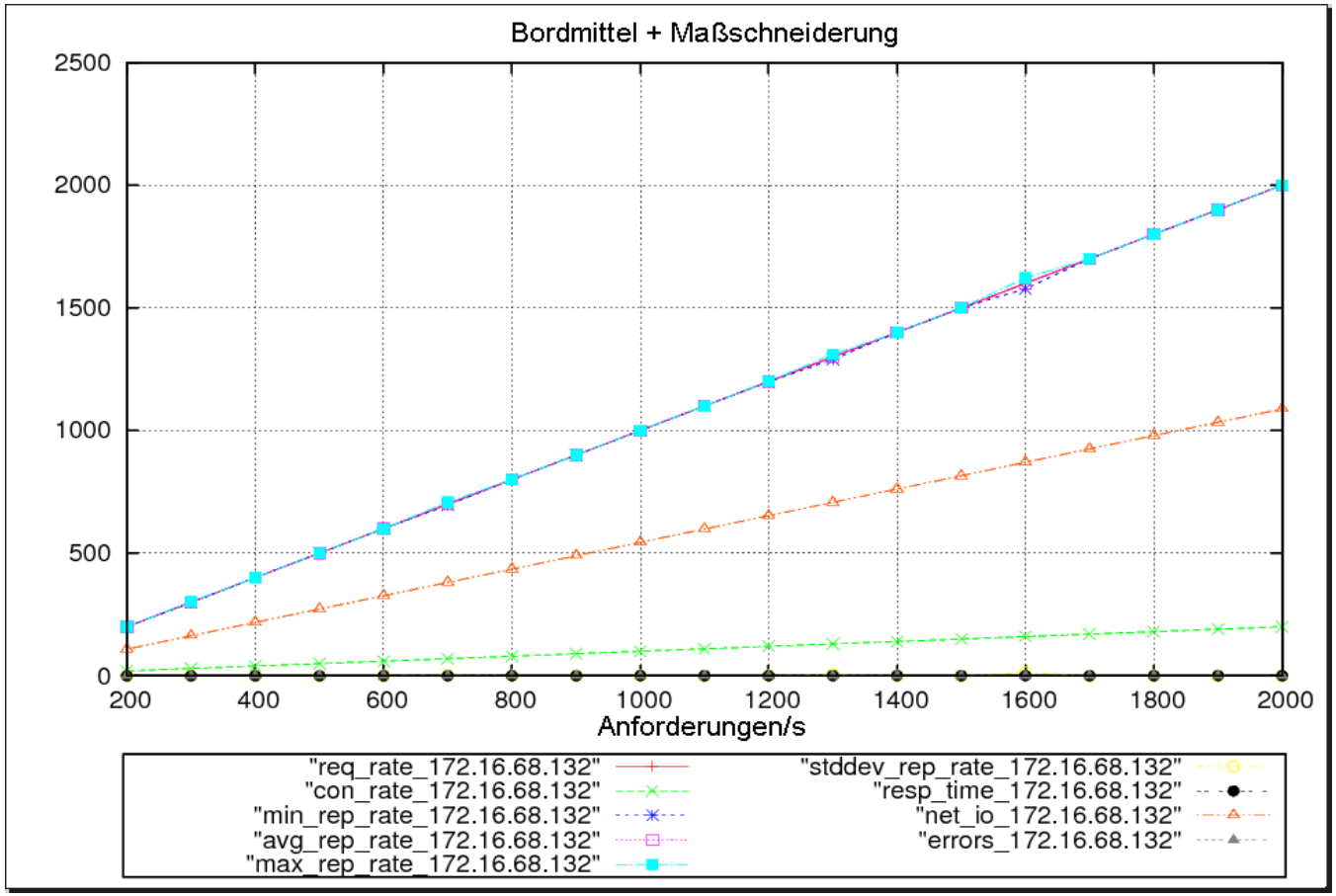
- IO-Scheduler CFQ



Evaluation

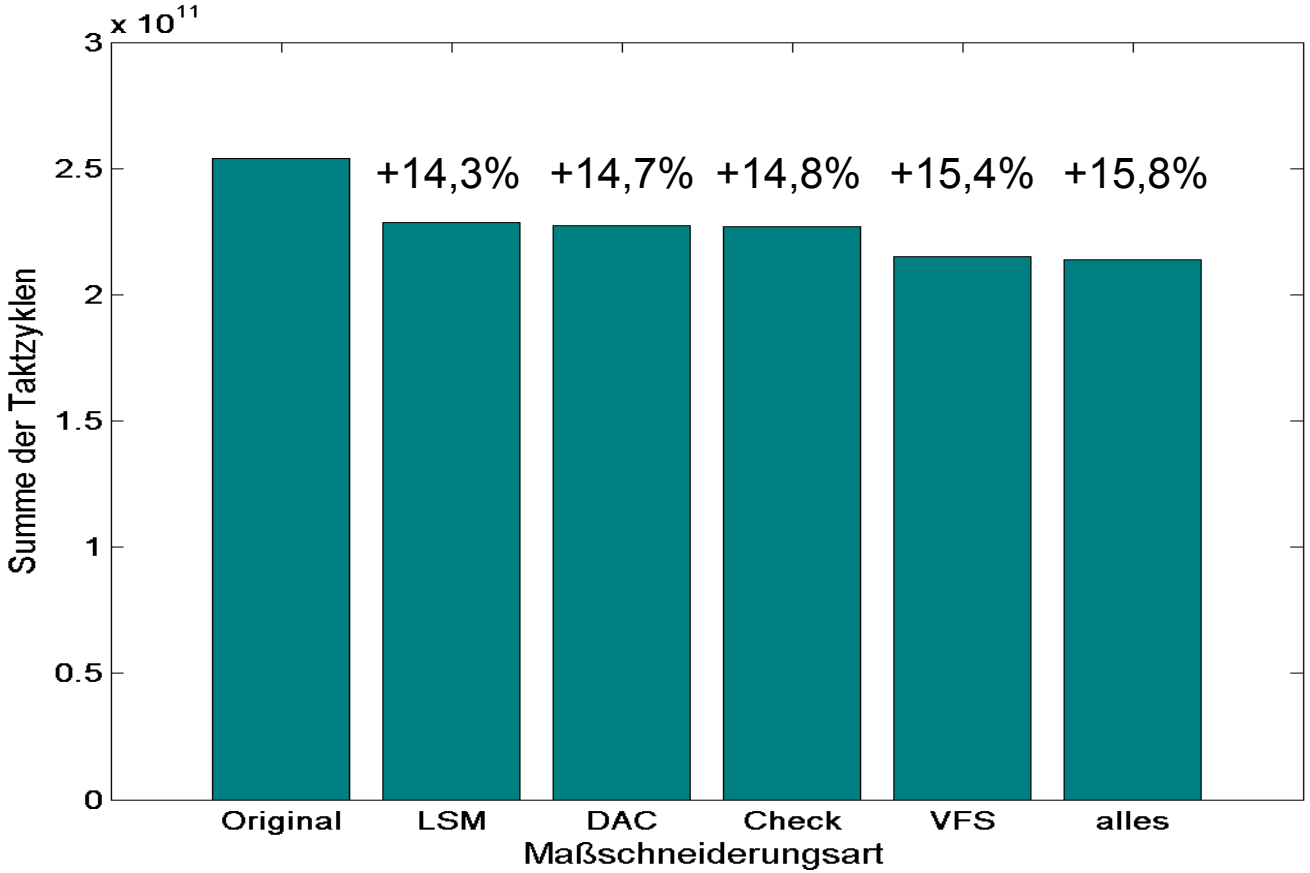
Auswirkungen der Bordmittel

- IO-Scheduler Deadline



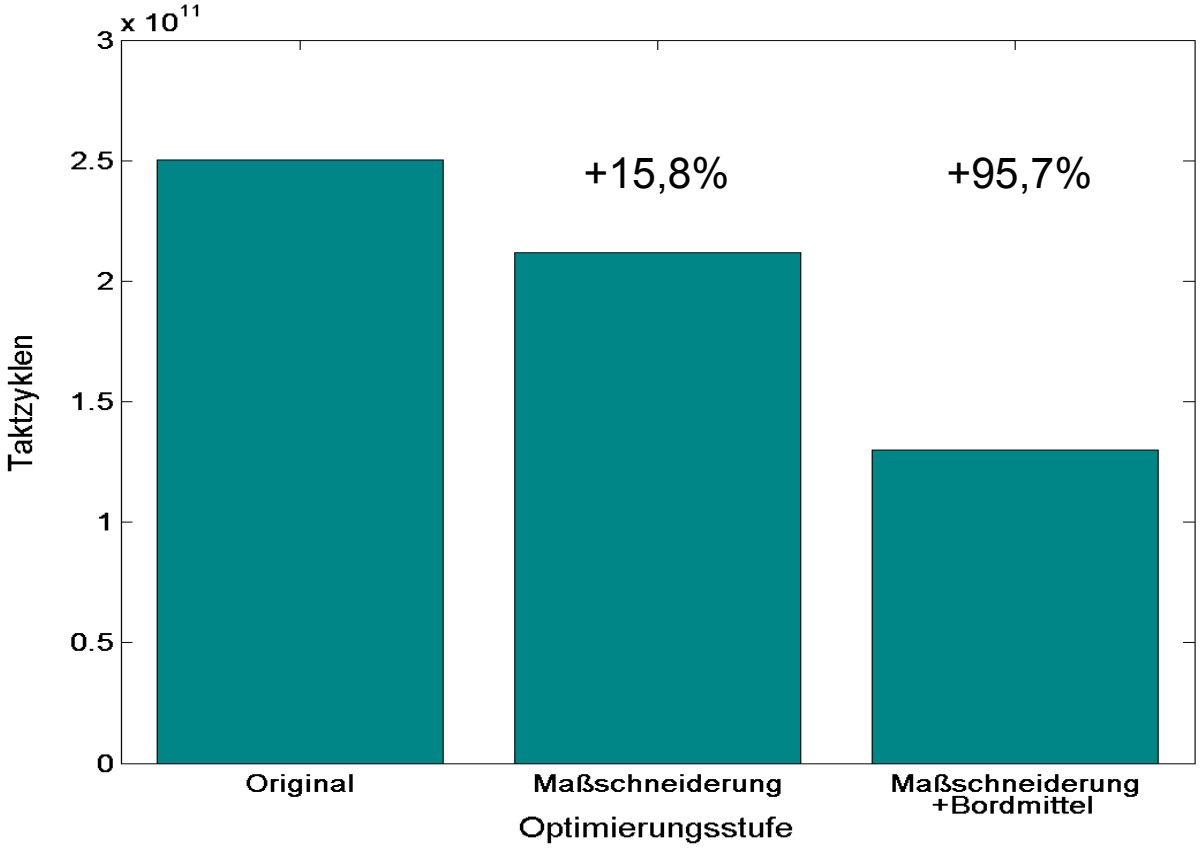
Evaluation

Auswirkungen der Maßschneiderung



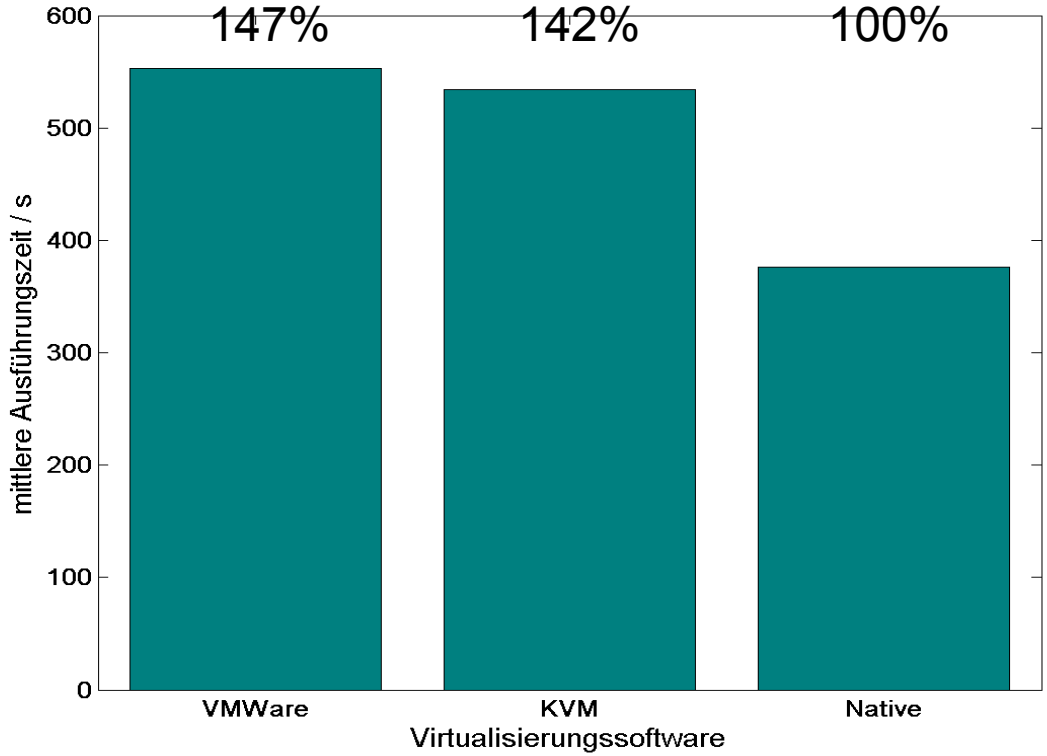
Evaluation

Maßschneiderung vs. Bordmittel



Evaluation

Auswirkungen eines Plattformwechsels (KVM, VMWare)

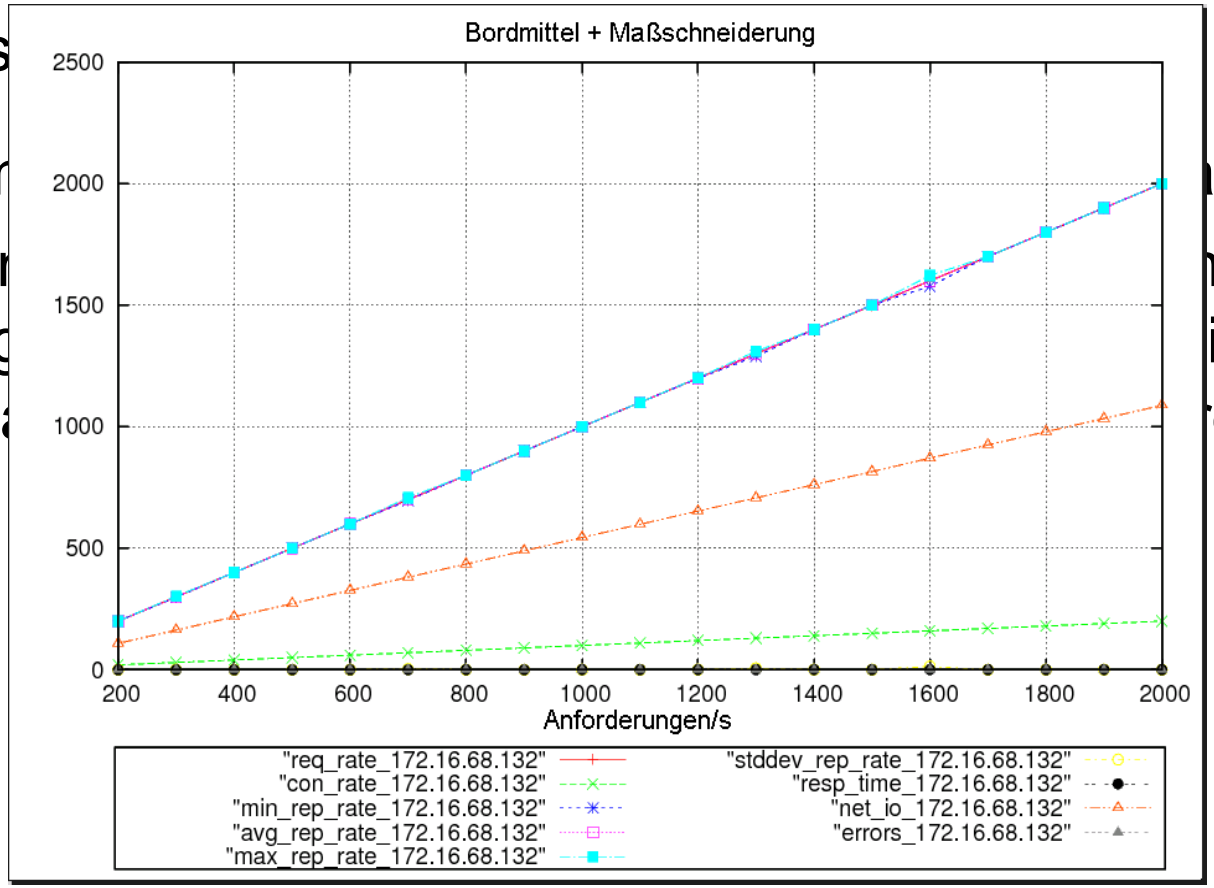


- VMWare mit Paravirtualisierung: 124,6%

Evaluation

Diskussion der Ergebnisse

- Maßs
- Für m tz nötig
  - Kern neide-
    rung itektur
  - ev. a re benutzen



Agenda

- Einführung
- Grundlagen
- Systemanalyse
- Implementierung
- Evaluation
- Zusammenfassung und Ausblick

Zusammenfassung und Ausblick

Maßschneiderung des Betriebssystems im Bereich virtueller Maschinen kann schnell durch andere Faktoren ausgebremst werden => ganzheitlicher Ansatz

- Einsatz des passenden I/O Schedulers verhinderte Einbruch der Antwortzeiten (vorher 500-1000, jetzt 2000 Anfragen pro Sekunde)
- Entfernen nicht benötigter Pakete +0,04%
- Alle Maßschneiderungen zusammen ergeben einen Gewinn von 15,8% der Gesamt-CPU-Zyklen
- Technische Anpassungen des Quellcodes durch den GCC brachten teilweise sogar Verschlechterungen

Zusammenfassung und Ausblick

- Je nach Virtualisierungsarchitektur können zusätzliche Performancegewinne realisiert werden (Overhead kann von 47% auf 24,6% gegenüber der Ausführungszeit auf der nativen Maschine schrumpfen)
- Selbst die Wahl der Virtualisierungslösung ist wichtig. KVM ist um 5% performanter als VMWare.

Insgesamt konnte eine Beschleunigung um 95,7% (gemessen in CPU-Zyklen) erreicht werden.

Zusammenfassung und Ausblick

Ausblick:

- Strukturen die Kmalloc() Aufrufe benutzen mit Slab-Caches verwalten
- Spezialisierer Tempo für quasi-konstante Parameterwerte benutzen
- Maßschneiderungspotenziale bei komplizierteren Virtual Appliances
- Auswirkungen von Updates der virtuellen Hardware

Fragen

Vielen Dank für Eure Aufmerksamkeit

Verwandte Arbeiten

Linux Kernel Specialization for Scientific Application Performance [MWK05]

- Linux wird sowohl für High-End Cluster als auch Batchbetrieb verwendet
  - Große Bemühungen um Optimierungen
- Anwendungen im Cluster sind ressourcenintensive Einzelanwendungen die exklusiven Zugriff auf die zugeteilte Maschine erlangen

Grundlagen

Verschiedene Granularitäten von Optimierungen

- Pakete (nicht genutzte Pakete entfernen)
- Module (nicht genutzte Module entladen, entfernen)
- Kernelkonfiguration (Kernel an Hardware exakt anpassen)
- Optimierungen der Syscalls im Kernel

Systemanalyse

- Benutzte Tools:
  - SystemTap:
    - Skriptsprache für Kprobes
    - Unterstützung um schnell und mit wenig Overhead Daten sammeln zu können
    - Messbar sind Callgraphen, Dauer, Häufigkeit, Scheduler-Entscheidungen, etc.
  - Ksplice
    - Erlaubt im laufenden Betrieb Änderungen am Kernel als Modul zu laden
    - Vorteile in Entwicklung, Schneller Wechsel von Änderungen am System möglich

Systemanalyse

- Simulieren von angemessenen Szenarien (z.B. bei lighttpd):
  - Peakperformance
    - Viele gleichzeitige Zugriffe auf beliebte Seite (z.B. index.html)
  - „Normales Surfverhalten“
    - Viele verschiedene Seitenaufrufe pro Webseitennutzer
    - Inkl. Downloads von größeren Nichttext-Dateien: z.B.: pdf
    - Mit Wartezeiten zwischen Aufrufen

Systemanalyse – Beispiel lighttpd

Gastsystem

- 1. Stap
  - Mitschneiden der Syscalls und speichern als Log

Unix-Timestamp + microsec | Task | Syscall | Parameterliste | Rückgabewert | Zeit im Kernel

```
1262593069485340 lighttpd poll(ufds=0x96f21c0 nfds=0x1 timeout_msecs=0x3e8 )=0 K
erneltime: 1001993
1262593069485375 lighttpd time(tloc=0x0 )=1262593069 Kerneltime: 6
1262593070487305 lighttpd poll(ufds=0x96f21c0 nfds=0x1 timeout_msecs=0x3e8 )=0 K
erneltime: 1001914
1262593070487336 lighttpd time(tloc=0x0 )=1262593070 Kerneltime: 5
1262593071489339 lighttpd poll(ufds=0x96f21c0 nfds=0x1 timeout_msecs=0x3e8 )=0 K
erneltime: 1001988
1262593071489372 lighttpd time(tloc=0x0 )=1262593071 Kerneltime: 5
1262593072491286 lighttpd poll(ufds=0x96f21c0 nfds=0x1 timeout_msecs=0x3e8 )=0 K
erneltime: 1001898
1262593072491299 lighttpd time(tloc=0x0 )=1262593072 Kerneltime: 4
1262593073493306 lighttpd poll(ufds=0x96f21c0 nfds=0x1 timeout_msecs=0x3e8 )=0 K
erneltime: 1002004
1262593073493337 lighttpd time(tloc=0x0 )=1262593073 Kerneltime: 5
1262593074107334 init select(n=0xb inp=0xffffffffbfaaed34 outp=0x0 exp=0x0 tvp=0
xffffffffbfaaee64 )=0 Kerneltime: 5005887
1262593074107358 init time(tloc=0x0 )=1262593074 Kerneltime: 4
1262593074107385 init stat(filename=0x804eb2f statbuf=0xffffffffbfaaeb24 )=0 Ker
neltime: 14
1262593074107403 init fstat(fd=0xa statbuf=0xffffffffbfaaeb24 )=0 Kerneltime: 3
1262593074107414 init stat(filename=0x804eb2f statbuf=0xffffffffbfaaeb24 )=0 Ker
neltime: 4
```

Systemanalyse – Beispiel lighttpd

Gastsystem

- 2. Logfile auswerten
  - Wieviele und welche Tasks laufen?

```
 1 chmod
 2 chown
 3 cron
 4 dhclient3
 5 dhclient-script
 6 exim4
 7 hostname
 8 init
 9 lighttpd
10 mv
11 rm
12 rsyslogd
13 run-parts
~
~
~
~
~
~
~
~
~
~
~
"stress.tasks" 13L, 97C                                1,1           Alles
```

Systemanalyse – Beispiel lighttpd

Gastsystem

- 2. Logfile auswerten
  - Welche Syscalls werden dafür benötigt (Anwendung / System)?

```
 1 access
 2 alarm
 3 brk
 4 chdir
 5 clock_gettime
 6 close
 7 connect
 8 dup2
 9 execve
10 faccessat
11 fchmodat
12 fchownat
13 fcntl
14 fork
15 fstat
16 fstatat
17 futex
18 getcwd
19 getdents
20 getegid
21 geteuid
22 getgid
23 getgroups
24 getpgid
"stress.syscalls" 70L, 551C                    1,1           Anfang
```

Systemanalyse – Beispiel lighttpd

Gastsystem

- 2. Logfile auswerten
  - Wie lange laufen die Syscalls (best / avg. / worst case) ?

```
  1 9273 23
  2 12330 0
  3 12330 0
  4 12330 0
  5 12330 0
  6 33323 0
  7 33323 0
  8 34320 1
  9 34321 0
 10 40805 16
 11 42016 11
 12 42596 9
 13 43633 10
 14 45576 9
 15 46345 561
 16 46863 8
 17 47748 9
 18 48375 10
 19 49620 11
 20 50709 9
 21 52391 10
 22 52722 8
 23 54446 10
 24 55345 9
"stress.write.messwerte" 83255L, 1022307C                    1,1           Anfang
```

```
Auswertung für stress.write.messwerte

Min | Max | Avg
-------------------------
0 118937 116.63
```

Systemanalyse – Beispiel lighttpd

Gastsystem

- 2. Logfile auswerten
  - Interessante syscalls identifizieren und später genauer betrachten (Callgraph)

```
 1 close    89078
 2 fcntl    98422
 3 ioctl    168317
 4 open     81064
 5 poll     4712
 6 read     168317
 7 send     80865
 8 sendfile         80865
 9 setsockopt       161731
10 shutdown         7968
11 stat     324
12 time     4712
13 write    83234
14 writev   80866
```

Systemanalyse – Beispiel lighttpd

Gastsystem

- 4. Callgraphen für syscalls überprüfen
  - Welche Fälle treten auf?
  - Gibt es Stellen innerhalb des Syscalls die unnötig lange brauchen?

```
   0 lighttpd(2245):->sock_sendpage file=0xf3e676c0 page=0xc193b860 offset=0x0 size=0x69 ppos=0xffffffff6a23de8 more=0x0
   0 lighttpd(2245): ->kernel_sendmsg sock=0xf6cab680 msg=0xf6a23d40 vec=0xf6a23d5c num=0x1 size=0x69
   0 lighttpd(2245):  ->sock_sendmsg sock=0xf6cab680 msg=0xf6a23d40 size=0x69
   0 lighttpd(2245):  <-sock_sendmsg return=0x69
   0 lighttpd(2245): <-kernel_sendmsg return=0x69
   0 lighttpd(2245):<-sock_sendpage return=0x69
   0 lighttpd(2245):->sock_sendpage file=0xf3e676c0 page=0xc193b8c0 offset=0x0 size=0x1000 ppos=0xffffffff6a23de8 more=0x1
3928 lighttpd(2245): ->kernel_sendmsg sock=0xf6cab680 msg=0xf6a23d40 vec=0xf6a23d5c num=0x1 size=0x1000
4448 lighttpd(2245):  ->sock_sendmsg sock=0xf6cab680 msg=0xf6a23d40 size=0x1000
4969 lighttpd(2245):  <-sock_sendmsg return=0x1000
5026 lighttpd(2245): <-kernel_sendmsg return=0x1000
5028 lighttpd(2245):<-sock_sendpage return=0x1000
   0 lighttpd(2245):->sock_sendpage file=0xf3e676c0 page=0xc193b8e0 offset=0x0 size=0x6be ppos=0xffffffff6a23de8 more=0x0
   1 lighttpd(2245): ->kernel_sendmsg sock=0xf6cab680 msg=0xf6a23d40 vec=0xf6a23d5c num=0x1 size=0x6be
   1 lighttpd(2245):  ->sock_sendmsg sock=0xf6cab680 msg=0xf6a23d40 size=0x6be
 192 lighttpd(2245):  <-sock_sendmsg return=0x6be
 206 lighttpd(2245): <-kernel_sendmsg return=0x6be
 219 lighttpd(2245):<-sock_sendpage return=0x6be
   0 lighttpd(2245):->sock_sendpage file=0xf3e676c0 page=0xc193b900 offset=0x0 size=0x1000 ppos=0xffffffff6a23de8 more=0x1
  36 lighttpd(2245): ->kernel_sendmsg sock=0xf6cab680 msg=0xf6a23d40 vec=0xf6a23d5c num=0x1 size=0x1000
  59 lighttpd(2245):  ->sock_sendmsg sock=0xf6cab680 msg=0xf6a23d40 size=0x1000
 621 lighttpd(2245):  <-sock_sendmsg return=0x1000
 637 lighttpd(2245): <-kernel_sendmsg return=0x1000
 651 lighttpd(2245):<-sock sendpage return=0x1000
```