HEINRICH HEINE
UNIVERSITÄT
DÜSSELDORF

Abteilung Betriebssysteme
Betreuer: John Mehnert-Spahn
Gutachter:
Prof. Dr. Michael Schöttner
Prof. Dr. Stefan Conrad

Leistungsbewertung einer Linux-basierten Cluster Checkpointing-Impementierung

Präsentation der Bachelorarbeit von Eugen Feller

Inhaltsverzeichnis

Einleitung

Einleitung

- Checkpointing und Recovery:
  - System muss einen aktuellen, konsistenten und stabilen Zustand so oft wie möglich auf der Festplatte sichern
  - Im Fehlerfall wird der letzte, fehlerfreie Zustand wiederhergestellt
- Zwei Strategien:
  - Sequentielles Checkpointing
    - Erzeugung eines kompletten Abbildes einer Anwendung
  - Inkrementelles Checkpointing
    - Nur die Änderungen zum vorherigen Checkpoint werden gesichert
- Ziel dieser Bachelorarbeit:
  - Erweiterung des bestehenden, sequentiellen Checkpointers
    - Inkrementelle Sicherung der Daten ermöglichen
  - Leistungsbewertung beider Implementierungen

Grundlagen

Kerrighed

- Entstanden aus einem Forschungsprojekt, 1999
- Ziel: Cluster als ein einziges SMP-System darstellen
- Besteht aus einem Satz von im Kernel verteilten Diensten:
  - Globales Management von Ressourcen des Clusters
- Prozess Ghosting
  - Prozesszustandsinformationen werden extrahiert, gesichert (Festplatte, Netzwerk, Hauptspeicher) und zur Prozesswiederherstellung verwendet
- Container Mechanismus:
  - Austausch der Daten zwischen den Knoten unter Gewährleistung der Datenkonsistenz
- Kerrighed beinhaltet einen koordinierten, sequentiellen Single-Prozess-Checkpointer
  - Ermöglicht das Checkpointen und Restarten von Prozessen mit
    - Einem Thread
    - Einer Vater-Sohn-Beziehung

Ghost-Mechanismus

- Zentraler Mechanismus zur Datenübertragung zwischen den Knoten
- Eingesetzt an vielen Stellen in Kerrighed
  - Prozessmigration
  - Checkpoint und Restart
  - …
- Beim Checkpoint und Restart:
  - Erstellung eines Prozessabbildes im File-Ghost
  - Image wird mit den zur Verfügung stehenden Ghost-Operationen auf die Festplatte geschrieben
  - Wird beim Restart eines Prozesses ausgelesen
- Ghost-Operationen erlauben per Default nur einen sequentiellen Zugriff auf die Daten

Dateien des Kerrighed Checkpointers

- Kerrighed Checkpointer sichert mit einem Checkpoint Aufruf komplette Anwendungen
- Checkpoint Images werden in dem Ordner „/var/chkpt/“ festgehalten
- Folgenden Dateien werden mittels des Ghost-Mechanismus angelegt:

description_Vx.txt (ASCII)	Datum, Version, Beschreibung (optional)
global_vX.bin (Binär)	Application ID, Version, Knoten Maske
node_0_vX.bin (Binär)	Beschreibung der in einen Checkpoint Prozess involvierten, lokalen Aufgaben
task_pid_vX.bin (Binär)	Register, Stack, Signal Masken
task_mm_pid_vX.bin (Binär)	Virtuelle Adresse, Page Protection Flags, alle physikalischen Seiten

Strukturen des Kerrighed Checkpointers

- Jeder Eintrag der „task_mm_pid_vX.bin“-Datei besteht aus drei Elementen:

Virtuelle Adresse	Page Protection Flags	Seite
0x100	Page Protection Flags	Seite 0
…	…	…
0xXYZ	Page Protection Flags	Seite N
0	Anzahl der Seiten ->	(N+1)

- Eine abschließende 0 kennzeichnet das Ende der Datei
- Zusätzlich wird die Anzahl der exportierten Seiten festgehalten

Implementierung des inkrementellen Checkpointings

Grundidee

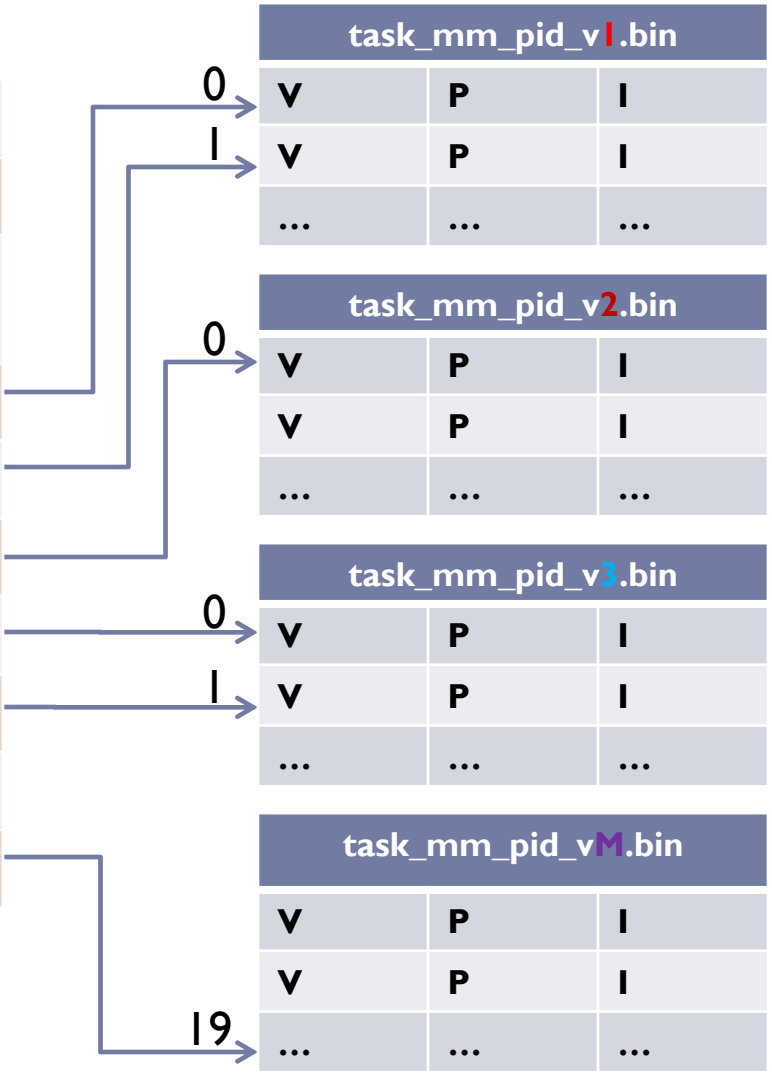
- Erweiterung des bestehenden Kerrighed Checkpointers
  - Ermöglichen einer inkrementellen Sicherung der Prozessdaten
- Dazu notwendig:
  - Entwicklung einer Verwaltungsstruktur zur Lokalisierung der veränderten Seiten
  - Abfangen von Schreibzugriffen und Aktualisieren der Verwaltungsstruktur
  - Einbettung der Verwaltungsstruktur in den bestehenden Checkpointer
  - Ermöglichen eines Restarts mit Hilfe der Verwaltungsstruktur

Verwaltungsstruktur für modifizierte Seiten

Verwaltungsstruktur (control_pid_vX.bin)

VMA_Count	Address_Count	Application_ID
5	20	0
Offset	Virtuelle Adresse	Version
0	0x100	1
1	0x200	1
0	0x300	2
0	0x400	3
1	0x500	3
…	…	…
19	0xXYZ	M

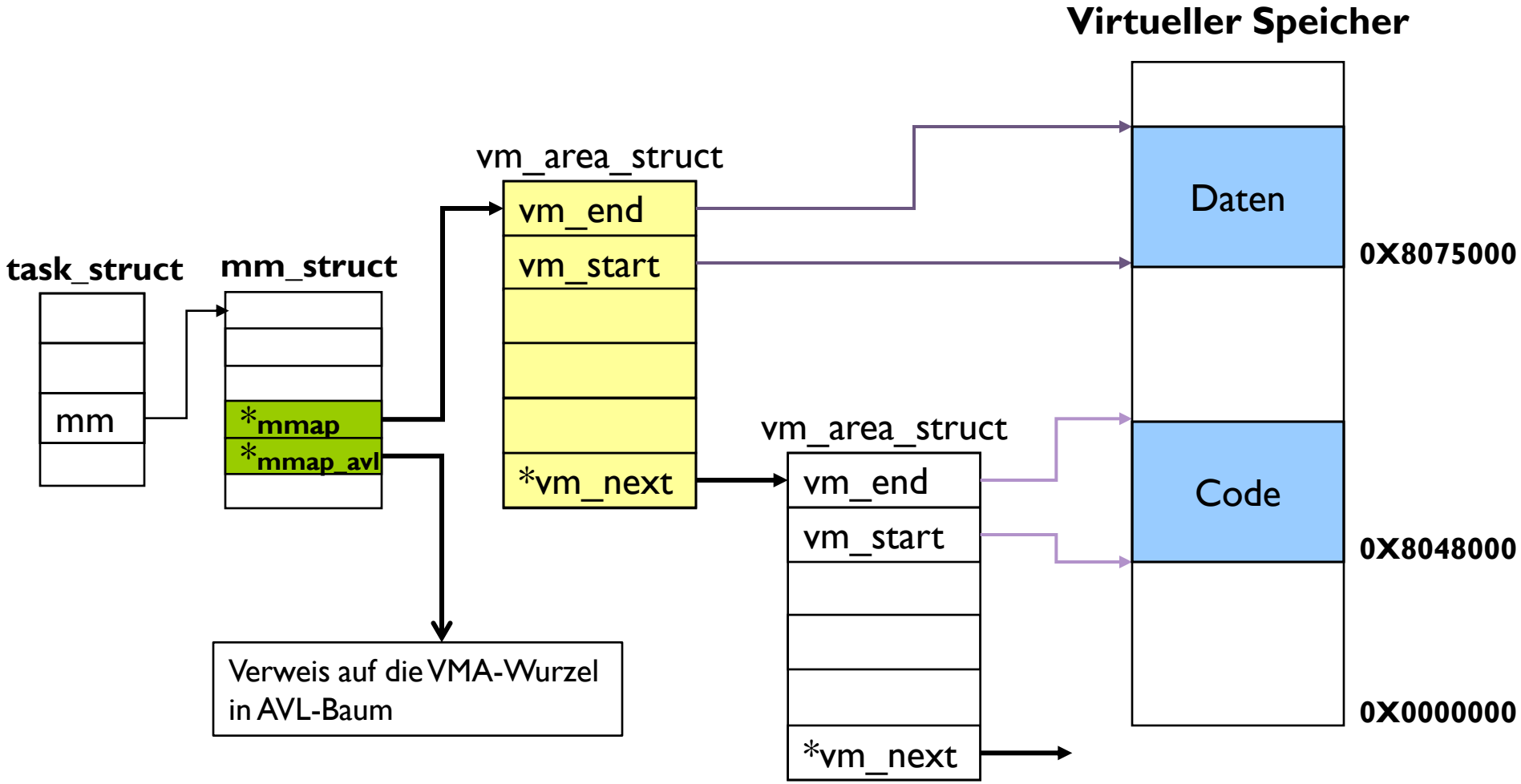
V := Virtuelle Adresse
P := Page Protection Flags
I := Physikalischer Seiteninhalt



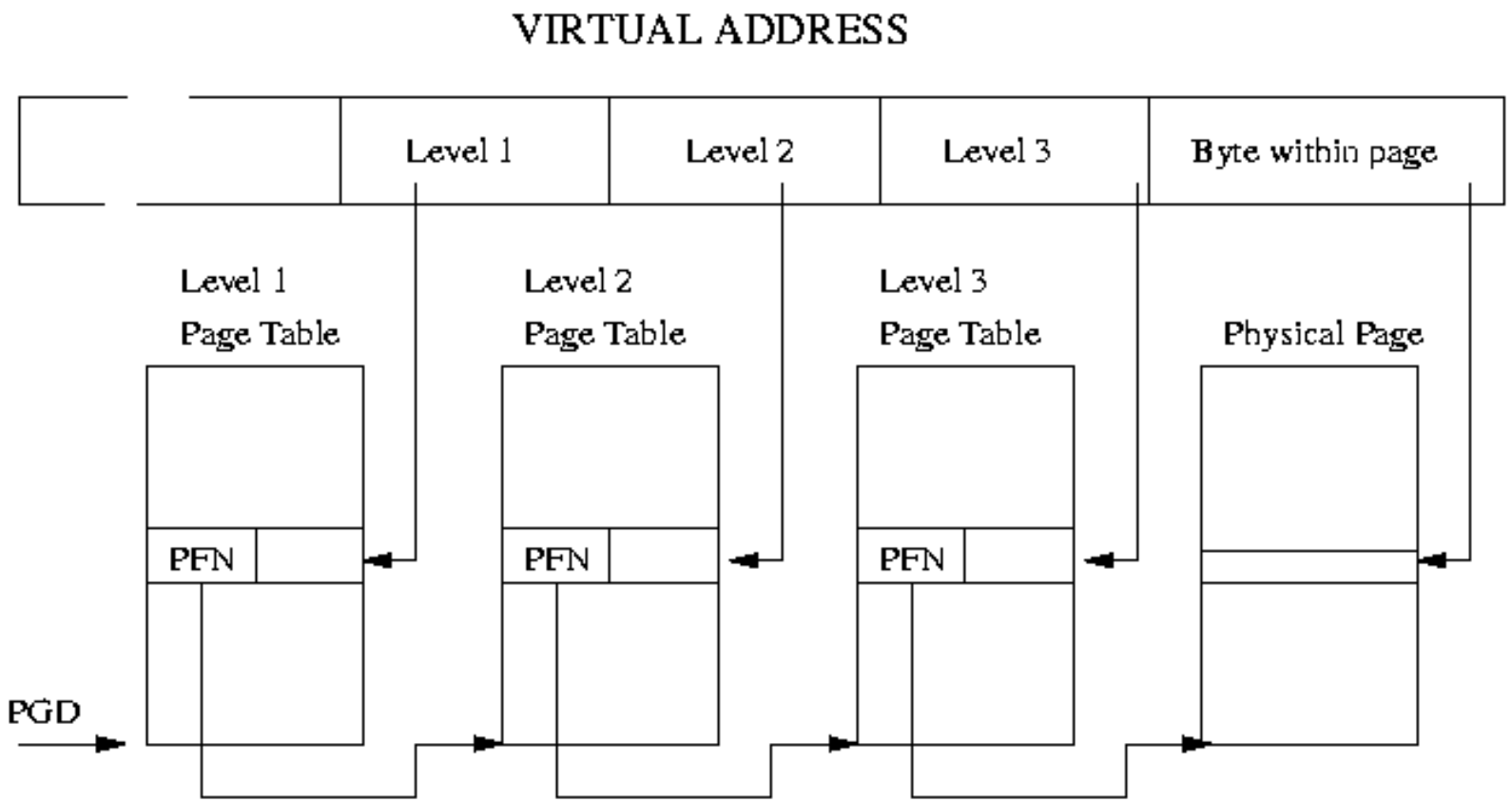
Änderung der VMAs und Adressen

- Verwaltungsstruktur der inkrementellen Sicherung erfasst nur die veränderten Seiten
  - Present und Write-Bit gesetzt
- Anzahl der VMAs und Adressen einer Anwendung kann sich zur Laufzeit ändern
  - Neue VMAs, die schreibgeschützte Seiten enthalten, kommen dazu
  - Schreibgeschützte Seiten werden einer bestehenden VMA hinzugefügt
  - Keine Erfassung innerhalb der Verwaltungsstruktur
    - Write-Bit nicht gesetzt
- Beide Fälle werden erkannt
  - Anzahl der VMAs und Adressen wird innerhalb der Verwaltungsstruktur festgehalten
  - Erstellung einer kompletten Sicherung
- Vorteile dieses Ansatzes:
  - Keine Aktualisierung der Verwaltungsstruktur bei einer Änderung der Seitenanzahl notwendig
  - Ermöglicht eine schnellere Sicherung der Anwendung

Repräsentation eines Prozesses auf der Kernel-Ebene



Repräsentation eines Prozesses auf der Kernel-Ebene



Repräsentation eines Prozesses auf der Kernel-Ebene

- Einträge der Page Table heißen Page Table Entries (PTEs)
- Beinhalten:
  - Page Frame Number (PFN)
  - Zugriffsschutzbits
- Zugriffsschutzbits:

Bezeichnung	Beschreibung
Dirty-Bit	Wird beim Schreibzugriff vom Prozessor gesetzt
Accessed-Bit	Wird bei einem beliebigen Zugriff gesetzt
Present-Bit	Gibt an ob die Seite im Speicher eingelagert wird
Read/Write-Bit	Lese-/Schreibzugriffsberechtigung
User/Supervisor-Bit	Speicherschutz

Initiale, inkrementelle Sicherung

- In der initialen, inkrementellen Sicherung
  - Erstellung der Verwaltungsstruktur
  - Offset wird mit einer 0 initialisiert und mit jeder gültigen virtuellen Adresse hochgezählt
  - Beinhaltet alle gültigen virtuellen Adressen mit einem Verweis auf die Version der Datei mit dem physikalischen Inhalt der Seiten
  - In einer zusätzlichen Struktur wird die Anzahl der VMAs und Adressen festgehalten
  - Abspeichern der Verwaltungsstruktur auf der Festplatte (control_pid_v1.bin)
  - Version der ersten Sicherung ist immer 1

Verwaltungsstruktur der initialen, inkrementellen Sicherung

Vma_Count	Address_Count	App_ID
5	20	0
Offset	Virtuelle Adresse	Version
0	0x100	1
1	0x200	1
2	0x300	1
3	0x400	1
4	0x500	1
…	…	…
19	0xXYZ	1

(control_pid_v1.bin)

Inkrementelle Sicherung

- In der nächsten, inkrementellen Sicherung:
  - Lade die vorherige Verwaltungsstruktur (control_pid_v1.bin)
  - Erkennung der veränderten physikalischen Seiteninhalte
  - Aktualisiere den Offset und die Version der virtuellen Adressen dessen physikalischer Inhalt verändert wurde
  - Aktualisierte Verwaltungsstruktur wird unter einer neuen Version (control_pid_v2.bin) gespeichert
    - Ermöglicht die Lokalisierung der physikalischen Seiten einer Anwendung zu einem beliebigen Zeitpunkt

Verwaltungsstruktur der inkrementellen Sicherung

Vma_Count	Address_Count	App_ID
5	20	0
Offset	Virtuelle Adresse	Version
0	0x100	1
1	0x200	1
2	0x300	1
0	0x400	2
1	0x500	2
…	…	…
19	0xXYZ	1

(control_pid_v2.bin)

Datenstruktur zur Realisierung der Verwaltungsstruktur

- Verwaltungsstruktur wurde unter Verwendung eines Red-Black-Trees (RBTs) realisiert
  - Binärer Suchbaum
  - Wird in der Linux Speicherverwaltung eingesetzt
  - Ermöglicht einen schnellen Zugriff auf die Elemente
    - Baum bleibt immer annähernd balanciert
  - Wichtige Operationen wie: *suchen, einfügen, löschen* in O(log n)
- Der Schlüssel jedes Knotens stellt eine virtuelle Adresse dar
- Im Kernel existiert ein Interface zum RBT
  - Erfordert die Implementierung der Such- und Einfüge-Operationen

Abfangen von Schreibzugriffen auf der Kernel-Ebene

- Inkrementelles Checkpointing:
  - Sicherung der seit dem letzten Checkpoint veränderten Daten
  - Erfordert eine Erkennung der Änderungen
- Jedes PTE beinhaltet einen Write-Bit
  - Zuständig für den Schreib/Lese-Zugriff
- Grundidee:
  - Für jede PTE einer gültigen, virtuellen Adresse, wird das Write-Bit zurückgesetzt
    - Anschließender Schreibzugriff bewirkt eine Page Fault Exception
      - Page Fault Handler des Kernels setzt das Write-Bit
  - Bei einem anschließenden inkrementellen Checkpoint kann das Write-Bit abgefragt werden
- Ist das Write-Bit gesetzt
  - Kennzeichnung einer Seitenmodifizierung

Abfangen von Schreibzugriffen auf der Kernel-Ebene

- Ablauf des inkrementellen Checkpointings (Einsammeln und Erkennen der veränderten Seiten):
  - Einen RBT (entweder neu oder aus der vorherigen Verwaltungsstruktur) erstellen
  - Doppelt verkettete Liste der VMAs durchgehen
  - Für jede gültige virtuelle Adresse
    - PTE holen und auf das PRESENT-Bit (und WRITE-Bit) prüfen
    - Den Knoten des Baumes (virtuelle Adresse, Offset, Version) erstellen (bzw. aktualisieren)
    - Den Inhalt der physikalische Seite holen und mittels des Ghost-Mechanismus auf der Festplatte sichern
    - Das Write-Bit zurück setzen
- RBT Speicherung in einer neuen Verwaltungsstruktur (control_pid_vX.bin)

Restart eines Prozesses mit Hilfe der Verwaltungsstruktur

- Strukturen wie Stack, Heap, Register, Prozessdeskriptor, Speicherdeskriptor, u.v.m bereits vom Kerrighed Checkpointer aufgebaut
- Doppelt verkettete Liste der VMAs aufgebaut
  - Inhalte der physikalischen Seiten müssen geladen werden
- Beim sequentiellen Restart
  - Sequentielles Auslesen der Inhalte (virtuelle Adresse, Page Protection Flags, Inhalt) aus der Datei „task_mm_pid_vX.bin“

Restart eines Prozesses mit Hilfe der Verwaltungsstruktur

- Beim inkrementellen Restart:
  - Inhalte der physikalischen Seiten auf mehrere Dateien (task_mm_pid_vX.bin) verteilt
  - Lokalisierung der Inhalte von den physikalischen Seiten notwendig
  - Verwaltungsstruktur kommt zum Einsatz
- Schritte beim inkrementellen Restart:
  - Verwaltungsstruktur laden
  - RBT aus der Verwaltungsstruktur generieren
  - Wiederherstellung des physikalischen Seiteninhalts mit Hilfe des RBT
    - Extrahieren der Page Protection Flags und der physikalischen Seiteninhalte.

Leistungsbewertung und Analyse (Sequentielles vs. Inkrementelles Checkpointing)

Ausgangssituation

- Gegenüberstellung der Performance
  - Sequentielles vs. Inkrementelles Checkpointing
- Konfiguration:

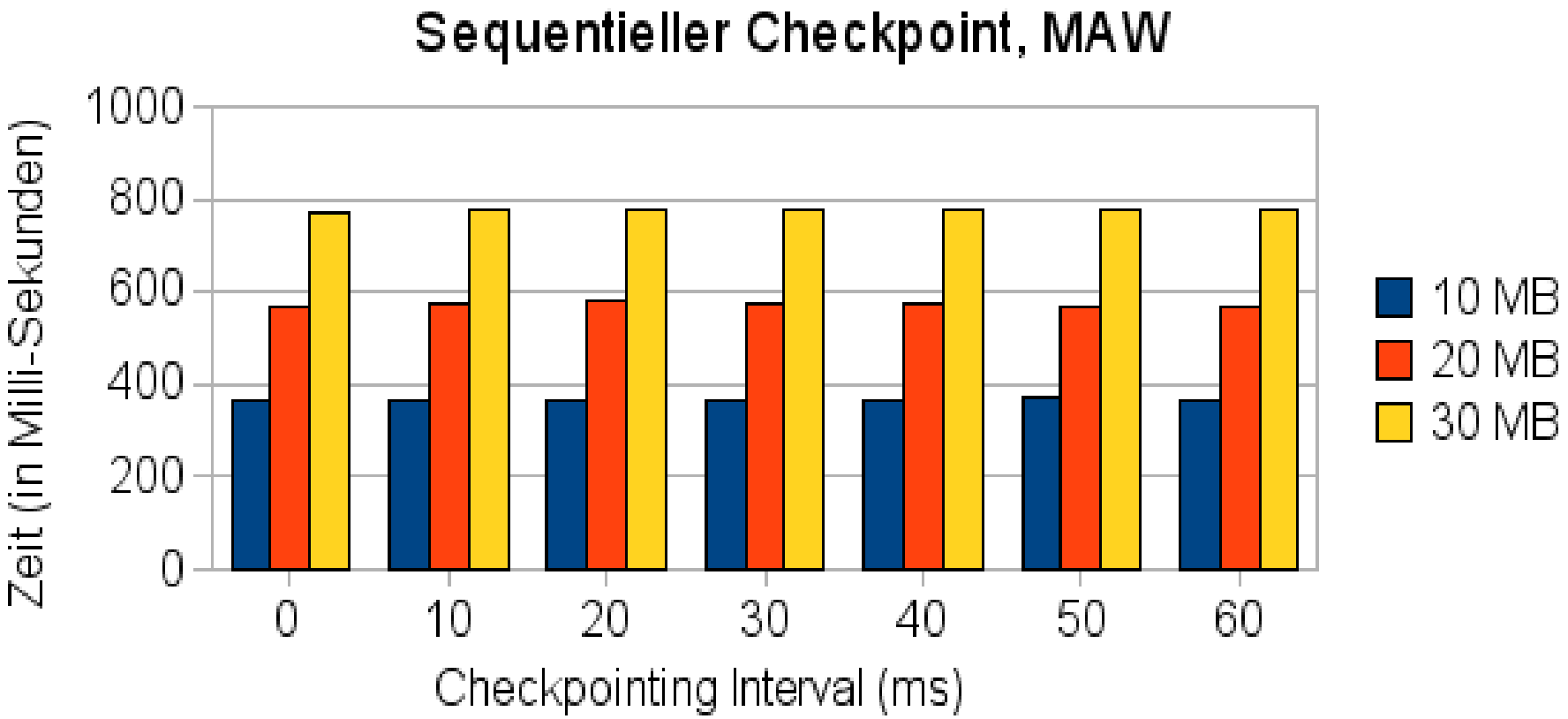
Prozessor (CPU)	AMD Opteron 244, 1.8Ghz
Arbeitsspeicher (RAM)	2GB
Auslagerungsspeicher (Swap)	2GB
Netzwerkkarte (Nic)	Gigabit
Kerrighed Version, LinuxSSI	3318

- Sicherungsintervall: 10ms
- Restart-Messungen sowohl mit als auch ohne Reboot des Systems

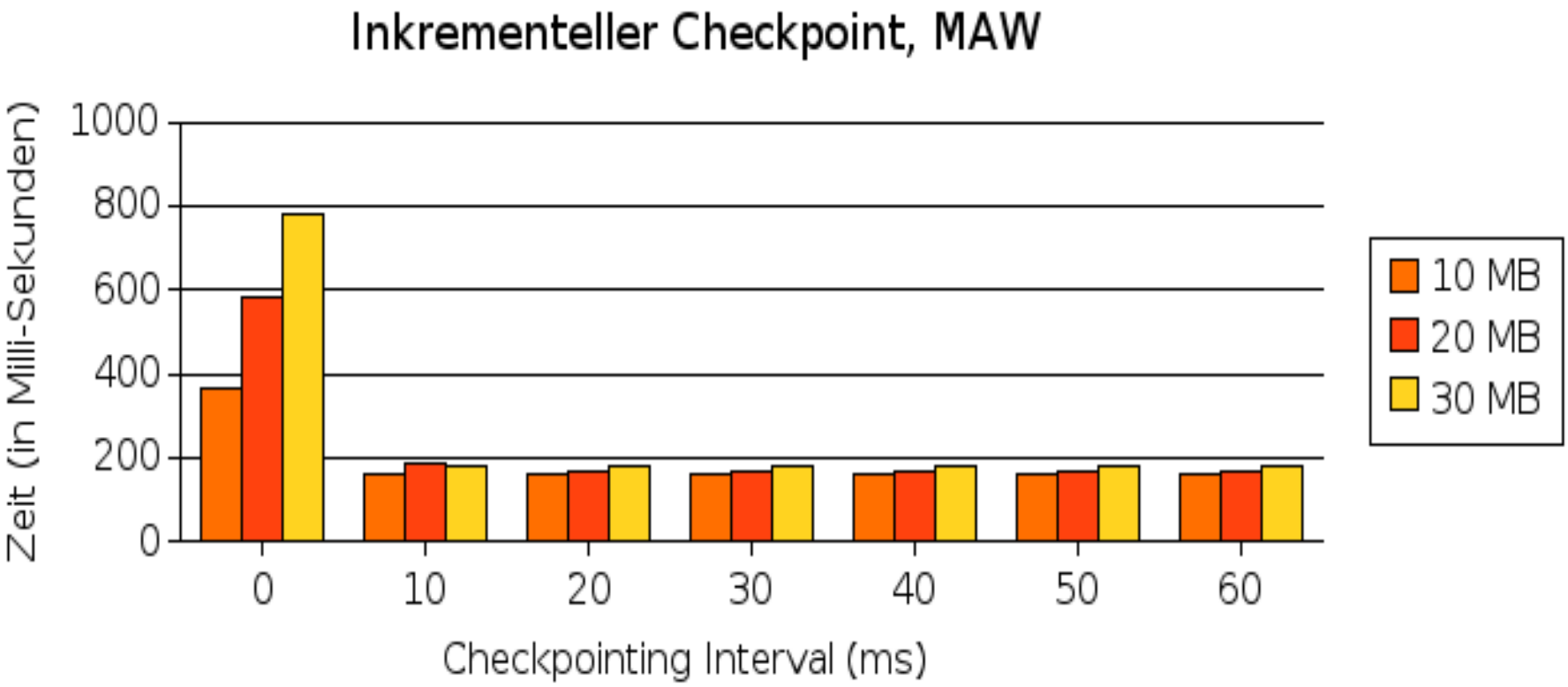
Messungen

- Messungen der Anwendung:
  - MallocAndWrite (MAW)
- MAW:
  - Alloziert n-MB Speicherplatz
  - Beschreibt den Speicherplatz komplett mit Zufallswerten
  - Anschließend in einer Endlosschleife
    - Beschreibung einiger Werte innerhalb des gegebenen Bereiches per Zufall
- Analyse weiterer Anwendungen mit anderen Zugriffsmustern

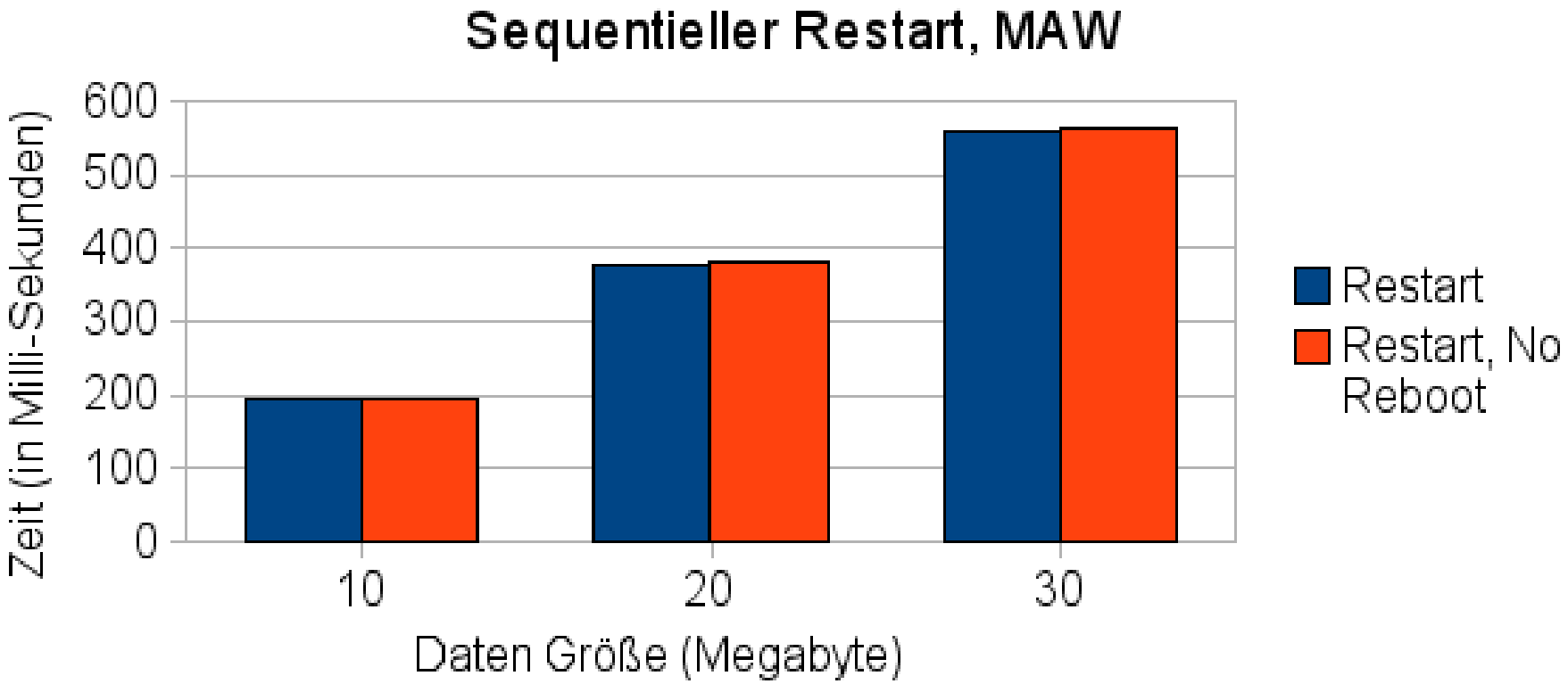
Messungen


Sequentieller Checkpoint, MAW
Zeit (in Milli-Sekunden)
1000
800
600
400
200
0
0
10
20
30
40
50
60
Checkpointing Interval (ms)
10 MB
20 MB
30 MB

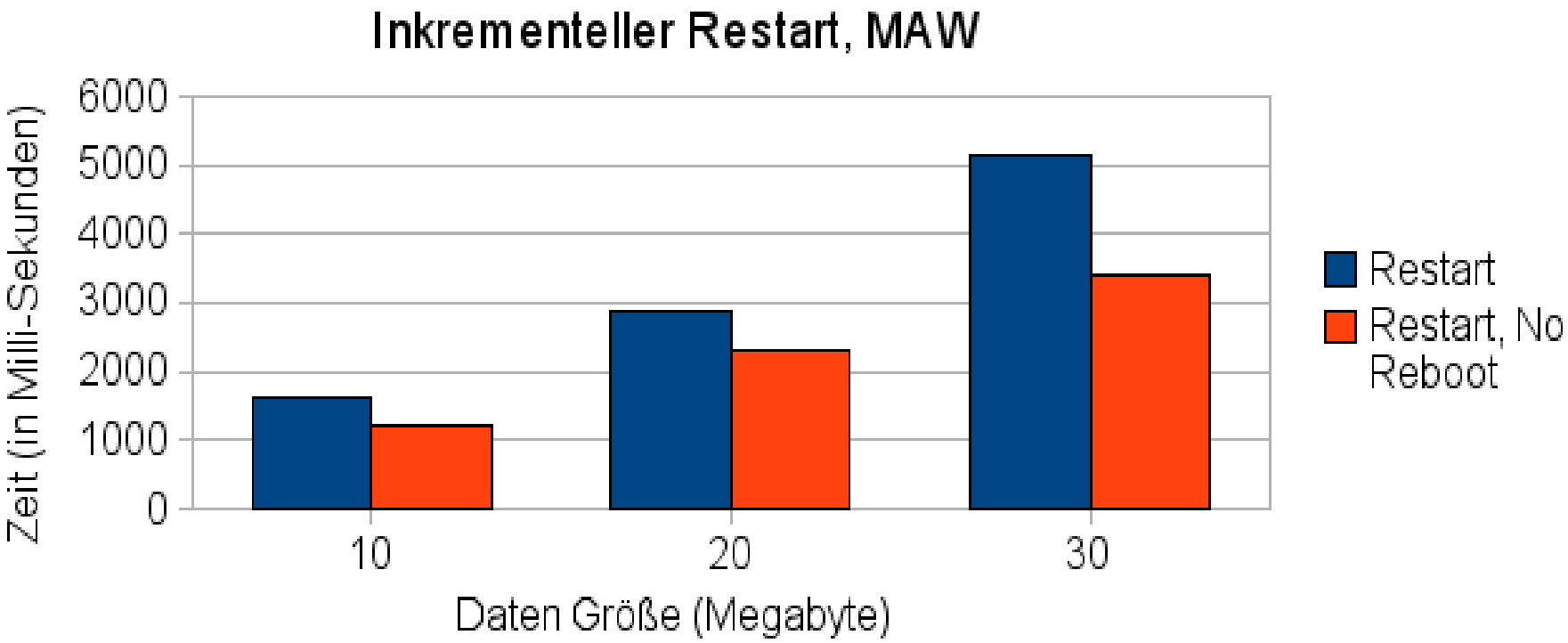
Messungen


Inkrementeller Checkpoint, MAW
Zeit (in Milli-Sekunden)
1000
800
600
400
200
0
0
10
20
30
40
50
60
Checkpointing Interval (ms)
10 MB
20 MB
30 MB

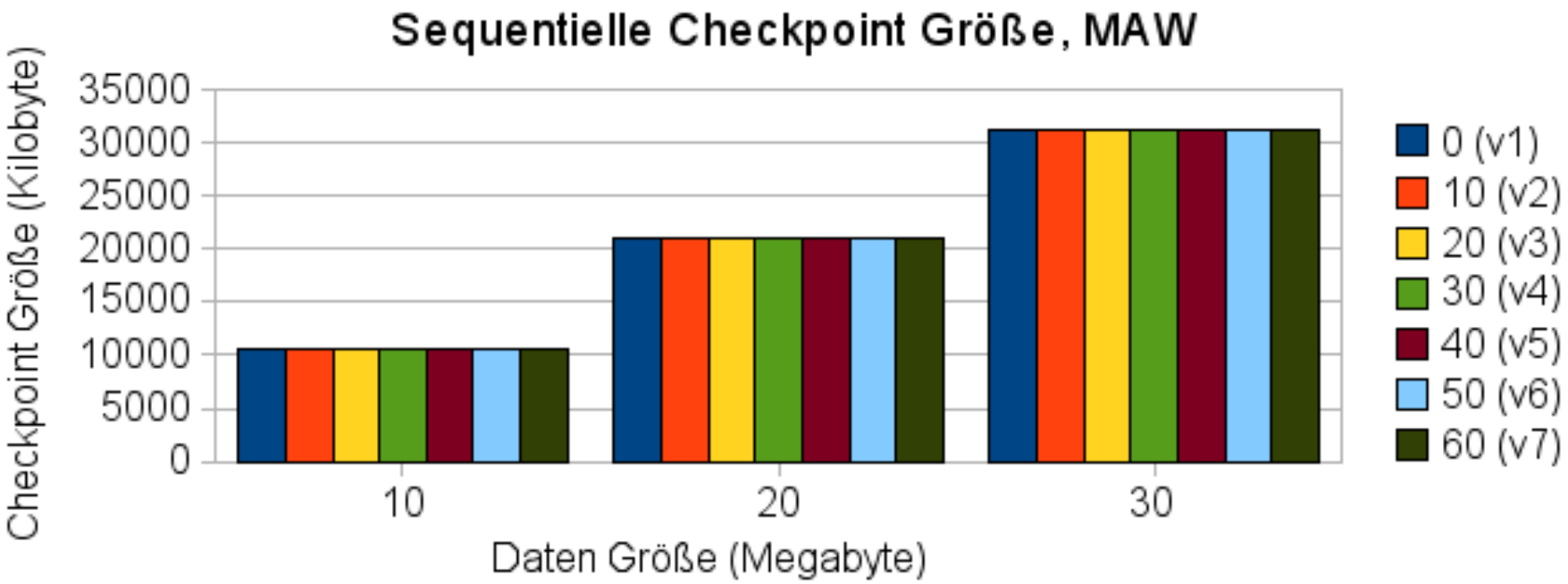
Messungen


Sequentieller Restart, MAW
Zeit (in Milli-Sekunden)
600
500
400
300
200
100
0
10
20
30
Daten Größe (Megabyte)
Restart
Restart, No Reboot

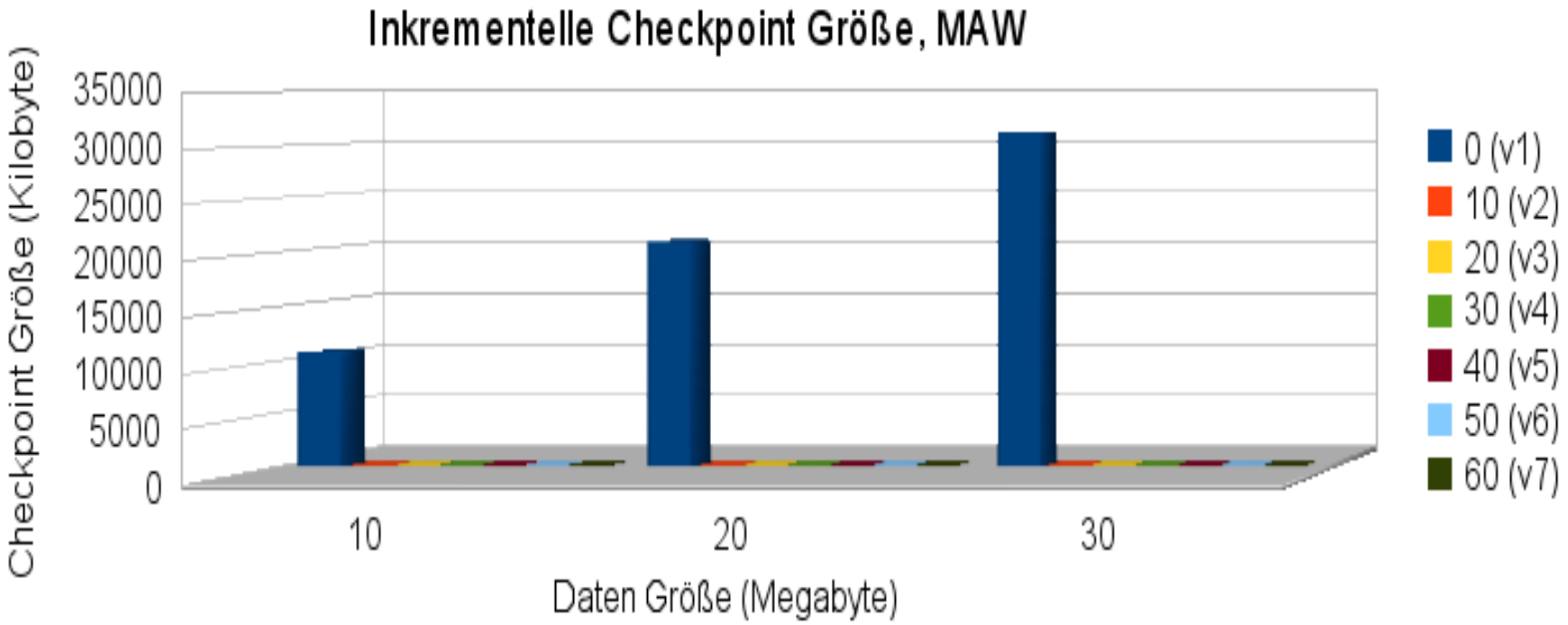
Messungen


Inkrementeller Restart, MAW
Zeit (in Milli-Sekunden)
6000
5000
4000
3000
2000
1000
0
10
20
30
Daten Größe (Megabyte)
Restart
Restart, No Reboot

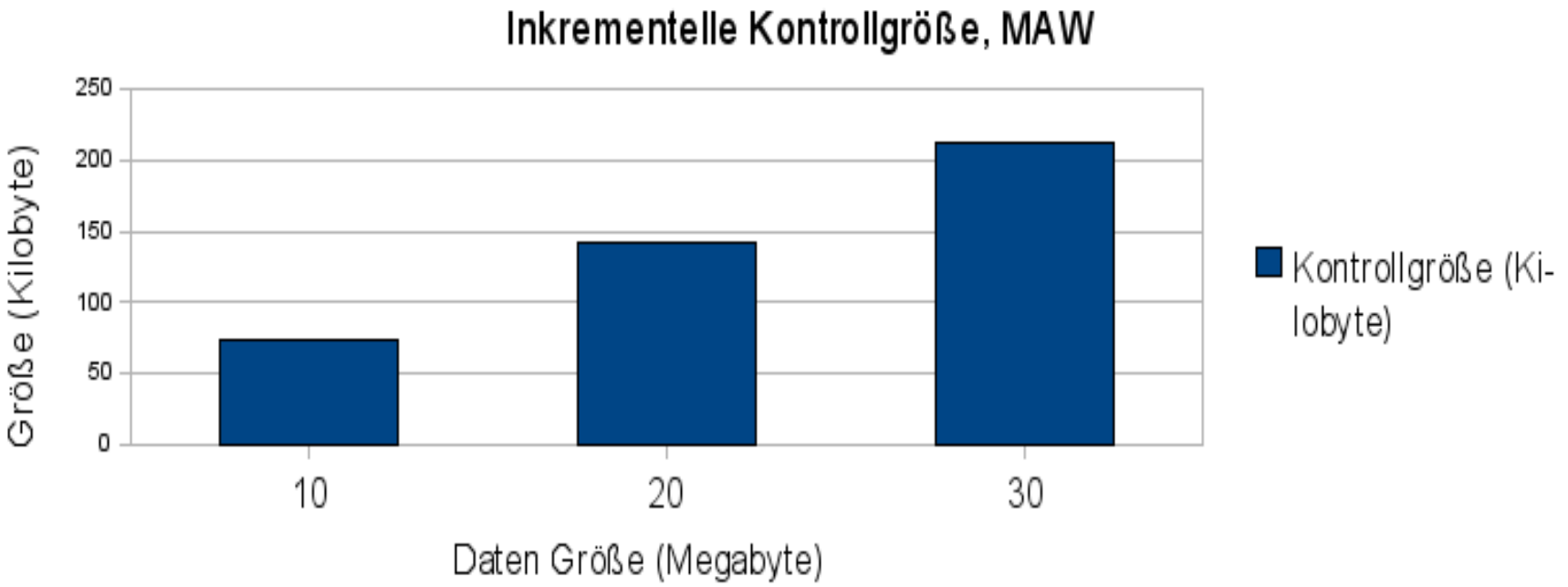
Messungen


Sequentielle Checkpoint Größe, MAW
Checkpoint Größe (Kilobyte)
35000
30000
25000
20000
15000
10000
5000
0
10
20
30
Daten Größe (Megabyte)
0 (v1)
10 (v2)
20 (v3)
30 (v4)
40 (v5)
50 (v6)
60 (v7)

Messungen


Inkrementelle Checkpoint Größe, MAW
Checkpoint Größe (Kilobyte)
35000
30000
25000
20000
15000
10000
5000
0
10
20
30
Daten Größe (Megabyte)
0 (v1)
10 (v2)
20 (v3)
30 (v4)
40 (v5)
50 (v6)
60 (v7)

Messungen


Inkrementelle Kontrollgröße, MAW
Größe (Kilobyte)
250
200
150
100
50
0
10
20
30
Daten Größe (Megabyte)
Kontrollgröße (Kilobyte)

Zusammenfassung und Ausblick

- Zusammenfassung
  - Konzept zum inkrementellen Checkpointing und Restart im Linux Kernel entwickelt
  - In den bestehenden Kerrighed Checkpointer eingebaut
  - Beide Checkpointing-Strategien auf ihre Performance vermessen und analysiert
    - Kleinere Checkpoint-Dateien
    - Schnellere Checkpointing Zeiten
    - Vorteile abhängig von der Anwendungsklasse
- Ausblick
  - Optimierung möglich
    - Seitentabellen ähnliche Verwaltungsstruktur
    - Anordnung der Dateistrukturen
  - Mit der Zeit wächst die Anzahl der Sicherungen
    - Garbage Collection notwendig

Live Demo
(Inkrementelles Checkpointing)

Vielen Dank für Ihre Aufmerksamkeit!