Leistungsanalyse

Hochleistungs-Ein-/Ausgabe

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Michael Kuhn

2019-05-21

Wissenschaftliches Rechnen
Fachbereich Informatik
Universität Hamburg

Leistungsanalyse

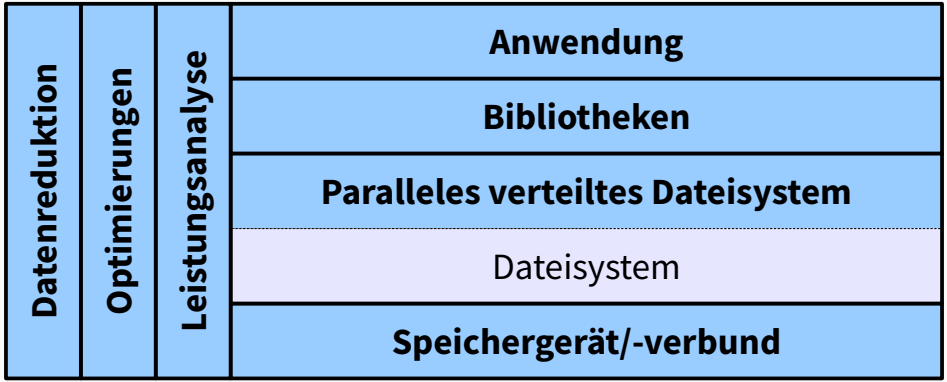


Abbildung 1: E/A-Schichten und orthogonale Themen

Einleitung

- Leistungsanalyse ist ein schwieriges Thema
  - Software und Hardware immer komplexer
  - Viele Schichten und Interaktionen
- Analyse besteht aus zwei Teilen
  - Messung und Bewertung
- Messung erfasst aktuelle Leistungsfähigkeit
  - Richtig messen ist auch schwierig
- Bewertung erlaubt Vergleich mit möglicher Leistung
  - Insbesondere bei Beschaffungen etc.

Einleitung...

- Messung
  - Wie messe ich?
  - Wie lange muss gemessen werden?
  - Wie oft muss gemessen werden?
  - Wie schließe ich äußere Einflüsse aus?
- Bewertung
  - Welche Leistung ist maximal möglich?
  - Welche Leistung kann ich erwarten?

Benchmarks

- Unmengen an Benchmarks für alle möglichen Aspekte verfügbar
  - CPU, Caches, RAM, Netzwerk etc.
- Für E/A ebenso viele Benchmarks verfügbar
  - IOzone, Bonnie, Bonnie++, PostMark, b_eff_io, FLASH I/O und viele mehr
  - Beispielhaft betrachtet: fio, IOR und mdtest
- Üblicherweise Erfassung eines bestimmten Zugriffsmusters
  - Daher viele verschiedene Benchmarks für viele verschiedene Anwendungsfälle

Beispiel: fio [2]

- Flexibler E/A-Tester
  - Autor ist Jens Axboe, Maintainer des Block-Layers
    - Unter anderem verantwortlich für die CFQ-, NOOP- und Deadline-Scheduler
    - Außerdem Entwickler des Werkzeugs `blktrace` und des System Calls `splice`
- fio kann beliebige Workloads messen
  - Häufig werden dafür viele spezielle Tests verwendet
- Unterstützung durch Job-Dateien
  - Gemeinsame und job-spezifische Parameter
  - Alles auch über Kommandozeile steuerbar
- Einschränkung: Parallelisierung nur über Threads

Beispiel: fio...

- Operationstypen
  - Lesen/schreiben/gemischt, sequentiell/zufällig
- Blockgröße
  - Sowohl einzelne Werte als auch Bereiche
- Datengröße
  - Wie viele Daten insgesamt zugegriffen werden sollen
- E/A-Engine
  - Synchron/asynchron, Memory Mapping, null
- Warteschlangentiefe
  - Für asynchrone E/A-Engines

Beispiel: fio...

- Buffered, direct, fsync
  - Zur Messung bzw. Umgehung des Caches
- Datei- und Threadanzahl
  - Parallele Workloads
- Sperren
  - Keine, exklusiv, nicht-exklusives Lesen
- Preallokation mit `fallocate`
  - Messung der Blockallokation
- Optimierung mit `fadvise`
  - Hinweise bezüglich Zugriffsmuster

- Ausrichtung
  - Beispielsweise zur Ausrichtung der E/A an Seitengrenzen
- Komprimierbar- und Deduplizierbarkeit
  - Aktuelle SSDs komprimieren die Daten transparent
  - Moderne Dateisysteme unterstützen auch Datenreduktion
- Durchsatzlimit
  - Nützlich um Hintergrundlast zu erzeugen
- Verifikation
  - Überprüfung ob die gelieferten Daten den vorher geschriebenen entsprechen

Beispiel: fio... [2]

```
1 [global]
2 rw=randread
3 size=128m
4
5 [job1]
6
7 [job2]
```

- Zufälliges Lesen aus 128 MiB großen Dateien
  - Werden automatisch angelegt
- Zwei Prozesse `job1` und `job2`
  - Dateinamen werden automatisch generiert
- Auch über Kommendozeile ausdrückbar
  - `fio --rw=randread --size=128m --name job1 --name job2`

Beispiel: fio... [2]

```
1 [random-writers]
2 ioengine=libaio
3 iodepth=4
4 rw=randwrite
5 blocksize=32k
6 direct=0
7 size=64m
8 numjobs=4
```

- Asynchrone E/A mit Tiefe 4
  - Immer vier asynchrone E/A-Operationen gleichzeitig
- Vier Prozesse, die zufällig gepuffert schreiben
  - Eigene, 64 MiB große Dateien mit einer Zugriffsgröße von 32 KiB
- `fio --name=random-writers ...`

Beispiel: fio...

- Unterstützt auch Trace-Replay
  - Erlaubt Ausführung aufgezeichneter E/A-Muster
  - Eigentliche Anwendung nicht notwendig
  - Vergleich mit anderen Systemen
- Interessantes Thema, da reale Anwendungen komplex sind
  - Viele Abhängigkeiten, schwierig zu kompilieren und auszuführen
- blktrace und eigenes Format
  - blktrace-Format ist binär
  - fio-Format ist Klartext und leicht selbst erstellbar
    - `write_iolog` und `read_iolog`

Beispiel: IOR [1]

- Für parallele verteilte Dateisysteme Zugriff von mehreren Knoten notwendig
  - fio erlaubt nur mehrere Prozesse auf einem Knoten
  - IOR nutzt MPI für parallelen Zugriff
- Unterstützung für mehrere Backends
  - Dummy, HDF5, HDFS, IME, mmap, MPI-IO, Parallel-NetCDF, POSIX, RADOS und S3
- Einige unterschiedliche E/A-Modi
  - Gemeinsame oder prozess-lokale Dateien
  - Prozessumordnung zur Umgehung von Cachingproblemen
    - Client X schreibt, Client X+n liest

Beispiel: IOR...

```
IOR START
  api=MPIIO
  testFile=/path/to/file
  repetitions=3
  readFile=0
  writeFile=1
  filePerProc=0
  keepFile=0
  segmentCount=10
  blockSize=1g
  transferSize=1m
RUN
IOR STOP
```

- Schreiben in gemeinsame Datei mit drei Wiederholungen
- Auch komplett über Kommandozeile steuerbar

Beispiel: IOR... [3]

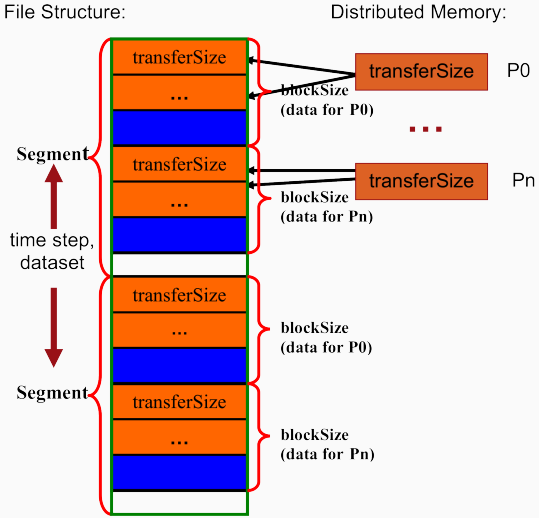


Fig. 1. The design of the IOR benchmark for shared file type. Blocks are stored in separate files for the 1-file-per-processor mode of operation.

Beispiel: mdtest

- Datendurchsatz durch Benchmarks gut abgedeckt
  - Metadaten aber auch ein wichtiger Faktor
- mdtest nutzt MPI für parallelen Metadatenzugriff
  - Operationen werden mit POSIX-Schnittstelle durchgeführt
  - MPI-IO bietet keine ausreichende Funktionalität
  - Neuerdings auch Unterstützung für IOR-Backends
- Aufgeteilt in drei Phasen
  - Erstellen, Status abrufen und Löschen
- Arbeitet mit hierarchischer Struktur

Beispiel: mdtest...

- Unterstützt ähnliche Funktionalitäten wie IOR
  - Prozessumordnung für Cache-Umgehung
  - Mehrere Wiederholungen
- Optional können Daten in Dateien geschrieben werden
  - Bei Bedarf kann auch synchronisiert werden
- Unterstützung für mehrere Wurzelverzeichnisse
  - Beispielsweise um mehrere Metadatenserver zu nutzen

Vergleich

- Vielzahl von Benchmarks für unterschiedlichste Anwendungsfälle
  - Sowohl Daten als auch Metadaten
- Ergebnisse allerdings nicht vergleichbar
  - Unterschiedliche Muster
  - Unterschiedliche Berechnung
  - Unterschiedliches Verhalten
- Unklarheiten bezüglich Ergebnissen
  - MB vs. MiB (Unterschied von $\approx 10\,\%$ bei TB/s)

Werkzeuge

- Benchmarks erlauben Erfassung der momentanen Leistung
  - Kein Aufschluss über Ursachen etc.
- Analyse und Optimierung erfordern Werkzeuge
  - Einblick in innere Vorgänge notwendig
  - Üblicherweise mit Hilfe von Tracing (Score-P)
- Manchmal abstrakte Betrachtung ausreichend
  - Grobe Übersicht über E/A-Verhalten (Darshan)

Beispiel: Darshan

- Werkzeug zur E/A-Charakterisierung
  - Sanskrit für „Sicht“ oder „Vision“
- Möglichst genaues Bild der Anwendungs-E/A
  - Einschließlich Eigenschaften wie der E/A-Muster
  - Dabei aber minimaler Overhead
- Geeignet für dauerhaften Einsatz
  - Wurde mit über 750.000 Kernen getestet
- Sehr gute Unterstützung für MPICH
  - Argonne National Laboratory
  - Gruppe um OrangeFS, MPICH und ROMIO

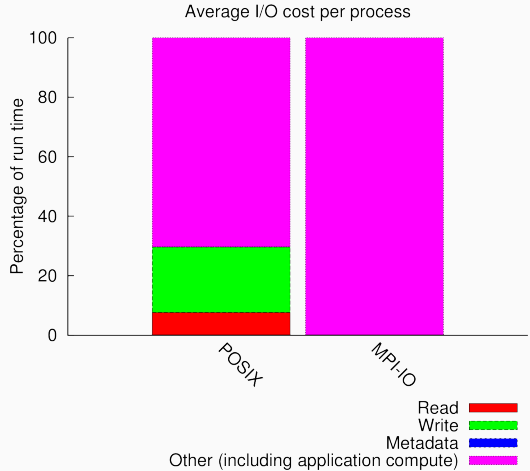
Beispiel: Darshan...

- Besteht aus zwei Teilen
  - Runtime und Werkzeuge
- Runtime erfasst Anwendungs-E/A
  - Spezifisch für eine MPI-Implementierung
  - Außerdem Optionen für Batch-Scheduler und gemeinsames Protokollverzeichnis
  - Compiler-Wrapper und Preload-Bibliothek `libdarshan.so`
- Werkzeuge analysieren aufgezeichnete Protokolle
  - `darshan-job-summary.pl`, `darshan-parser` etc.

Beispiel: Darshan...

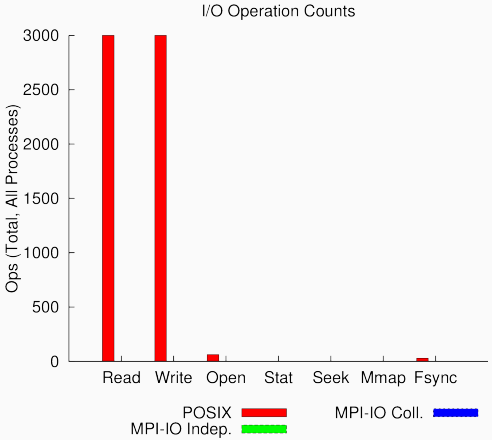
- MPI-parallelisierter POSIX-Benchmark mit zehn Prozessen
  - Erst Schreibphase, dann Lesephase
    - Durch Barriers getrennt
  - Blockgröße von 1 MiB
    - 100 Blöcke
  - Leeren des Caches zwischen Schreib- und Lesephase
    - `echo 3 > /proc/sys/vm/drop_caches`
  - `fsync` beim Schließen
  - Drei Wiederholungen

Beispiel: Darshan...



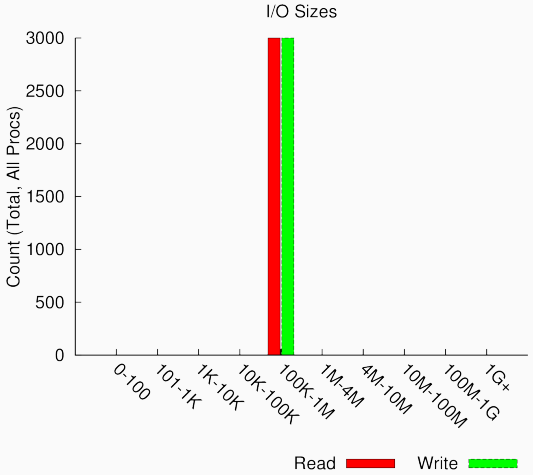
- Keine Berechnungen, trotzdem sehr hoher Other-Anteil

Beispiel: Darshan...



- Wie erwartet 3.000 Schreib- und Leseoperationen
  - 10 Prozesse $\times$ 100 Operationen $\times$ 3 Wiederholungen

Beispiel: Darshan...

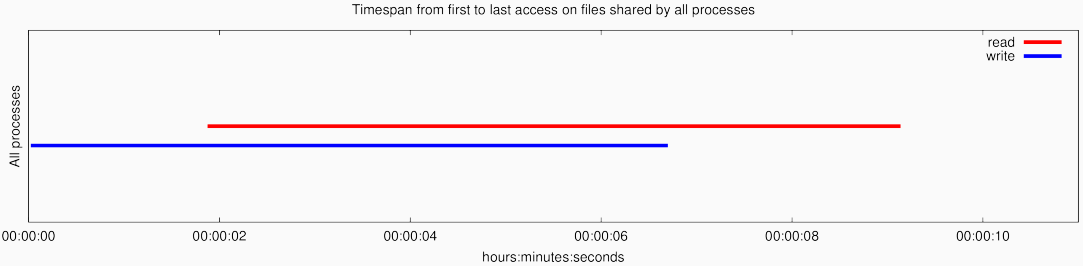


- Alle Operationen haben 1 MiB Größe

Beispiel: Darshan...

- Sequential: Zugriffe mit aufsteigendem Offset
- Consecutive: Zugriffe direkt hintereinander

Beispiel: Darshan...



- Anzeige durch drei Wiederholungen trügerisch
- Grobe Übersichten der Kosten von E/A
  - Bezüglich Aufrufanzahl, Zugriffsgröße und -muster
- Erlaubt Abschätzung ob Optimierung notwendig ist
  - Häufig genauere Analyse notwendig
  - Neuer Modus: Darshan eXtended Tracing (DxT)

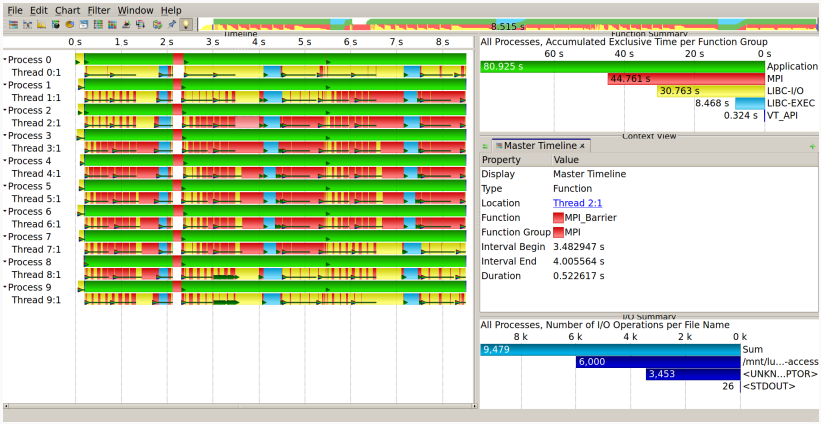
Beispiel: Score-P

- Score-P zeichnet Spurdaten auf
  - Score-P ist Open Source
  - Vorgänger: VampirTrace
  - Spezifisch für eine MPI-Implementierung
  - Compiler-Wrapper `scorep`
- Vampir zeigt Spurdaten an
  - Vampir ist kommerziell
  - Evaluationslizenzen verfügbar
- Spurdaten sind deutlich größer als Darshan-Protokolle
  - Im getesteten Fall mehr als Faktor 100

Beispiel: Score-P...

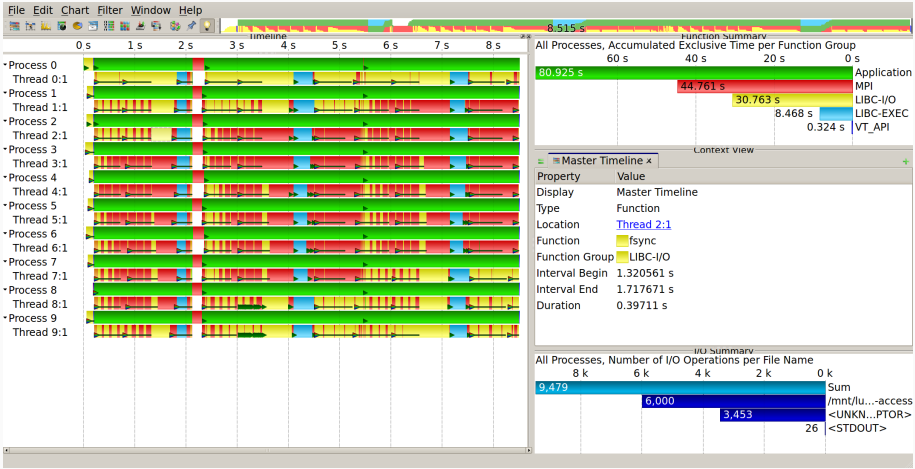
- Tracing erlaubt detaillierten Einblick in Anwendung und E/A
  - Standardmäßig werden Informationen zu jeder Operation aufgezeichnet
- Tracing erzeugt signifikanten Overhead
  - Laufzeit deutlich höher
  - Eventuell abweichendes Laufzeitverhalten
- Momentan noch keine E/A-Unterstützung
  - Dafür noch VampirTrace notwendig

Beispiel: Score-P...



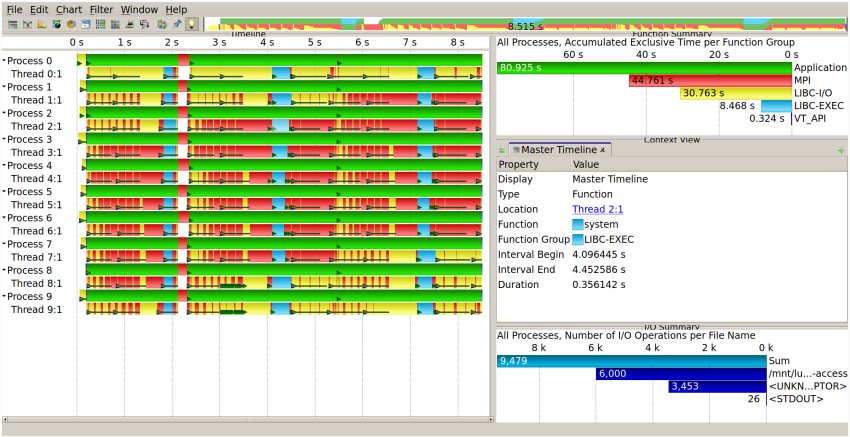
- Hauptthreads schlafen durchgängig
- Sehr ungleiche E/A-Zeiten, dadurch lange Barrieren

Beispiel: Score-P...



- Synchronisieren dauert teilweise sehr lange

Beispiel: Score-P...



- Leeren des Caches auch sichtbar
  - Blockiert bis Blöcke geschrieben sind

- Bewertung der Leistung durch Modellierung der theoretisch möglichen Leistung
  - Vergleiche Rmax und Rpeak auf der TOP500-Liste
- Dazu sind einige Informationen notwendig
  - Involvierte Komponenten
  - Leistungscharakteristika der Komponenten
- Zusätzliche Leistungsmessungen der Komponenten
  - Dazu wieder andere Werkzeuge

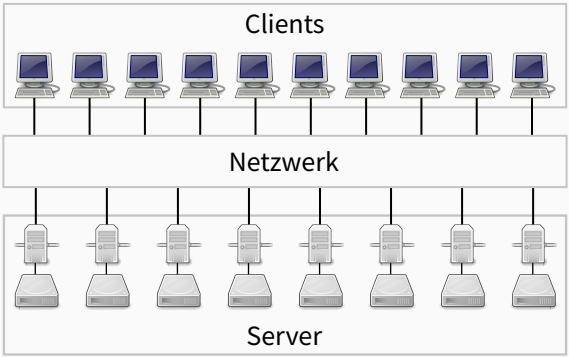
Beispiel

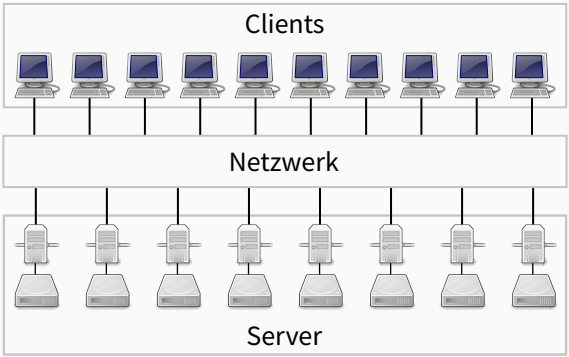
- Beispiel: Echtzeit-Twitter-Analyse
  - System kann in Echtzeit den Inhalt von Tweets untersuchen
  - Alternativ Tweets speichern und später analysieren
- Frage: Wie ist die Leistung zu bewerten?

Beispiel

- Beispiel: Echtzeit-Twitter-Analyse
  - System kann in Echtzeit den Inhalt von Tweets untersuchen
  - Alternativ Tweets speichern und später analysieren
- Frage: Wie ist die Leistung zu bewerten?
  - Ca. 6.000 Tweets pro Sekunde
  - 300 Bytes pro Tweet entspricht 1,8 MB/s
  - 51,6 TB pro Jahr

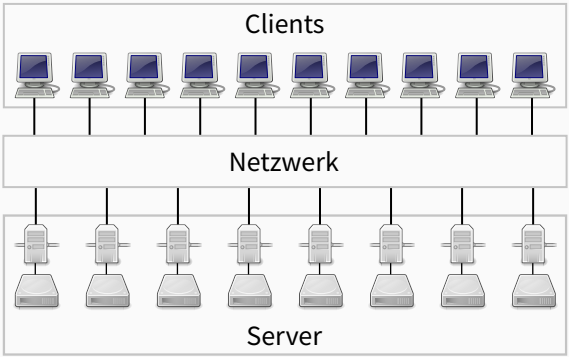
Übersicht





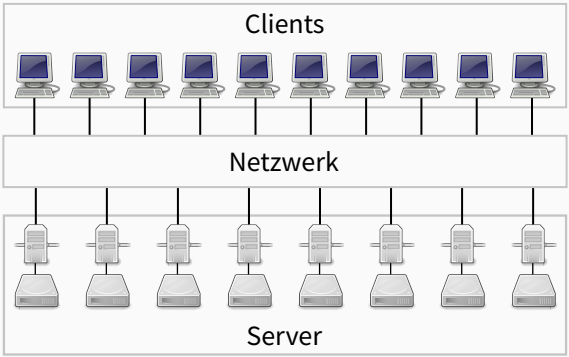
- Clients: IOPS, RAM, Netzwerkanbindung

Übersicht



- Clients: IOPS, RAM, Netzwerkanbindung
- Netzwerk: Durchsatz und Latenz

Übersicht



- Clients: IOPS, RAM, Netzwerkanbindung
- Netzwerk: Durchsatz und Latenz
- Server: Durchsatz und IOPS der Speichergeräte, Bus

- Anzahl der E/A-Operationen pro Sekunde
  - Kontextwechsel könnten Beschränkung sein
- Durchsatz und Latenz des Hauptspeichers
  - Üblicherweise kein Problem
- Abschätzung mit Hilfe von `tmpfs` und `fio`
  - Idee: Viele kleine E/A-Operationen ausführen

```
1 $ mkdir /tmp/fs
2 $ mount -t tmpfs tmpfs /tmp/fs
3 $ ...
4 $ umount /tmp/fs
```

Client...

```
$ fio --name=cs --filename=/tmp/fs/foo --rw=write --bs=1 \
      --size=1g --runtime=60 [--numjobs=n]
$ vmstat 1
$ fio --name=bw --filename=/tmp/fs/foo --rw=write --bs=1m \
      --size=5g --runtime=60
```

- Messung auf `west`-Knoten
- ≈ 1.000.000 IOPS
  - Blockgröße von 1 ist wichtig, da 0 eventuell durch libc behandelt
- ≈ 330.000 Kontextwechsel
- ≈ 4 GiB/s Durchsatz
  - Hauptspeicher üblicherweise höher, da Overhead durch `tmpfs`

Netzwerk

- Deutliche Unterschiede je nach Netzwerktechnologie
  - InfiniBand vs. Ethernet
- Durchsatz des Netzwerks
  - Sollte höher sein als Leistung der Speichergeräte
- Anzahl der Pakete pro Sekunde
  - Beschränkung bei vielen kleinen Nachrichten
  - Wichtig bei Metadatenoperationen
- Messung mit Hilfe von `ping` und `iperf`

Netzwerk...

```
1 $ ping -c 10000 -f $host
2 $ iperf --server --port $port
3 $ iperf --client $host --port $port
```

- Messung zwischen `west`- und `sandy`-Knoten
- Round Trip Time $\approx$ 0,100 ms
- Durchsatz $\approx$ 110 MiB/s

Netzwerk...

```
1 $ ping -c 10000 -f $host
2 $ iperf --server --port $port
3 $ iperf --client $host --port $port
```

- Messung zwischen `west`- und `sandy`-Knoten
  - Zwischen `west`-Knoten höhere Latenz
- Round Trip Time $\approx$ 0,100 ms
  - Entspricht $\approx$ 10.000 Nachrichten pro Sekunde
- Durchsatz $\approx$ 110 MiB/s
  - Entspricht Ethernet mit 1 Gbit/s

Speichergerät

- Deutliche Unterschiede je nach Speichertechnologie
  - HDD vs. SSD
- Durchsatz der Speichergeräte
  - Niedriger als Netzwerkdurchsatz für maximale Ausnutzung
  - Höher als Netzwerkdurchsatz für Leistungsreserven
- IOPS wichtig für Metadatenoperationen
  - Auch für zufällige Datenzugriffe
- Speicherbus kann auch beschränkender Faktor sein
  - Teilweise immer noch SATA 2.0 (300 MB/s)

Speichergerät...

```
$ fio --name=iops --filename=/dev/sd? --direct=1 --rw=randread --bs=4k \
      --size=$size --runtime=60
$ fio --name=bw   --filename=/dev/sd? --direct=1 --rw=read     --bs=1m \
      --size=$size --runtime=60
```

- Ungepufferte E/A, um Geräte zu messen
  - Schließt Einflüsse des Page Caches aus
- HDDs
  - IOPS $\approx$ 60–80
  - Durchsatz $\approx$ 120 MiB/s
- SSDs
  - IOPS $\approx$ 15.000
    - Ausreißer mit $\approx$ 5.500 (möglicherweise Garbage Collection o. Ä.)
  - Durchsatz $\approx$ 270 MiB/s

Beispiel

- Messung des Datendurchsatzes
  - Lustre und OrangeFS
- Unterschiedliche Blockgrößen
  - 1 MiB und 64 KiB
  - Entspricht den Streifenbreiten von Lustre und OrangeFS
- Bei 10.000 Nachrichten pro Sekunde
  - 9,8 GiB/s (1 MiB) bzw. 625 MiB/s (64 KiB) pro Knoten
- Flaschenhals ist Netzwerkdurchsatz
  - Maximal 1.100 MiB/s

Beispiel...

Dateisystem	Blockgröße	1 PPN	6 PPN	12 PPN
Lustre	1 MiB	640 MiB/s	105 MiB/s	110 MiB/s
OrangeFS	1 MiB	160 MiB/s	390 MiB/s	430 MiB/s
OrangeFS	64 KiB	250 MiB/s	115 MiB/s	180 MiB/s

Tabelle 1: Schreiben

Dateisystem	Blockgröße	1 PPN	6 PPN	12 PPN
Lustre	1 MiB	1.095 MiB/s	735 MiB/s	875 MiB/s
OrangeFS	1 MiB	130 MiB/s	265 MiB/s	430 MiB/s
OrangeFS	64 KiB	505 MiB/s	140 MiB/s	195 MiB/s

Tabelle 2: Lesen

- Massiver Leistungseinbruch bei Lustre
  - Beim Schreiben kein exklusiver Zugriff auf OST
- Streifenbreite als Blockgröße teilweise vorteilhaft

Beispiel...

Dateisystem	Blockgröße	1 PPN	4 PPN	8 PPN
Lustre	1/PPN MiB	640 MiB/s	620 MiB/s	605 MiB/s
OrangeFS	64/PPN KiB	250 MiB/s	280 MiB/s	210 MiB/s

Tabelle 3: Schreiben

Dateisystem	Blockgröße	1 PPN	4 PPN	8 PPN
Lustre	1/PPN MiB	1.095 MiB/s	1.800 MiB/s	525 MiB/s
OrangeFS	64/PPN KiB	505 MiB/s	655 MiB/s	455 MiB/s

Tabelle 4: Lesen

- Bessere Leistung in beiden Fällen
  - Insbesondere beim Schreiben auf Lustre und Lesen von OrangeFS
- Anomalie beim Lesen von Lustre
  - Muss durch Cache verursacht sein

Zusammenfassung

- Benchmarks erlauben die Erfassung der aktuellen Leistung
  - Viele unterschiedliche Benchmarks für viele unterschiedliche Anwendungsfälle
- Werkzeuge sind notwendig zur eingehenden Analyse
  - Sowohl für Grobüberblick als auch zur Detailanalyse
- Messung sagt nichts über mögliche Leistungsfähigkeit aus
  - Dafür ist eine Bewertung der Leistung notwendig
  - Häufig durch Leistungsmodellierung

Zusammenfassung...

- Grobe Modellierung ist häufig schon ausreichend
  - Lässt sich bei Bedarf verfeinern
- Unvorhersehbares Verhalten macht Analyse schwierig
  - Siehe beispielsweise Leseleistung bei Lustre
- Analyse der konkreten Implementierung notwendig
  - Optimierung setzt viele Detailkenntnisse voraus

Quellen

[1] high performance computing. **IOR - Parallel filesystem I/O benchmark.** `https://github.com/hpc/ior`.

[2] Jens Axboe. **fio - Flexible IO Tester.** `http://git.kernel.dk/?p=fio.git;a=summary`.

[3] Hongzhang Shan and John Shalf. **Using IOR to Analyze the I/O Performance for HPC Platforms.** In *In: Cray User Group Conference (CUG'07*, 2007.