Simulation paralleler E/A auf Anwendungs- und Systemebene

Julian M. Kunkel

Institut für Informatik
Parallele und Verteilte Systeme
Ruprecht-Karls-Universität Heidelberg

28.09.2009

Agenda

1. Ziele
2. Hardware Model
3. Software Model
4. Implementierung
5. Ergebnisse
6. Fazit

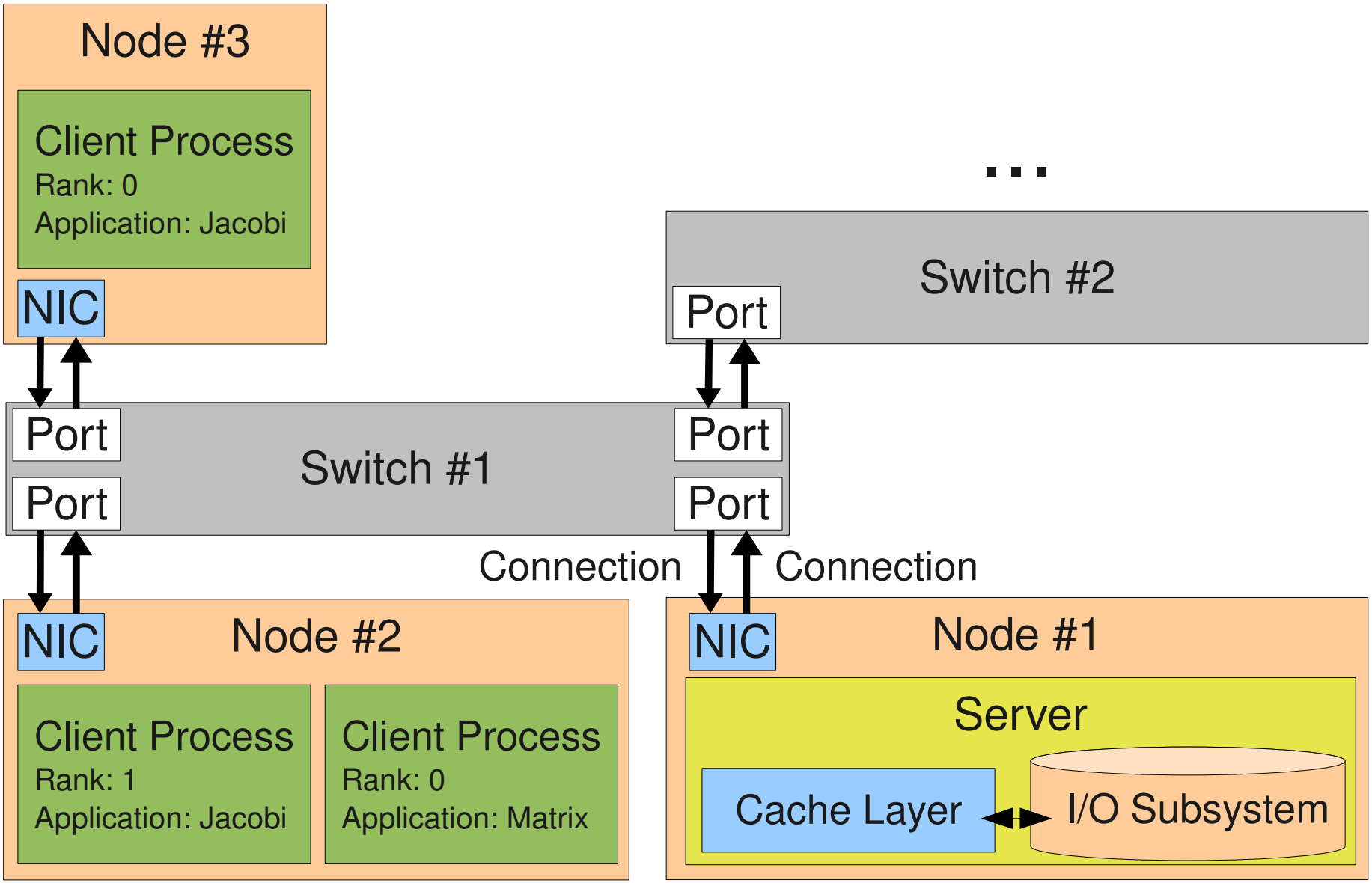
Ziele

- MPI and MPI-IO Befehle simulieren
  - Abschätzung für Effizienz einer Implementierung
- Einfache Modelle für Hardware und Software
- Konfiguration der Hardware/Software nach belieben
  - Schätze Skalierbarkeit von Algorithmus in beliebigen Cluster Umgebungen
- Einsatz/Nutzbarmachung von Standard-Tools zur Analyse
  - Simulationsergebnisse genau wie realle Programläufe bewerten
- Neue Algorithmen/Verhalten schnell und reproduzierbar testen
- Einsetzbarkeit für Lehre
  - Soll auf Desktop PC lauffähig sein

Hardware Modell

- Komponententypen an existierende Hardware angelehnt
  z.B. “Knoten” oder “I/O-subsystem”
- Je Komponententyp sind alternative Implementierungen möglich
  z.B. SSD oder Disk
- Cluster Modell beschreibt konkrete Implementierungen der Komponenten und Charakteristika
- Deterministisches Komponentenverhalten

Simulierte Komponenten



Hardware Model - Charakteristika

- Knoten:
  - # CPUs, Abarbeitungsgeschwindigkeit
- Netzwerkkarte:
  - Latenz, Durchsatz
- Server Cache Layer:
  - Cache Größe, Algorithmus
- I/O Subsystem:
  - Letzter Zugriff (Datei, Offset) bestimmt Zugriffszeit
  - Track-To-Track Zugriffzeit, Mittlere Zugriffszeit
- Switch:
  - Store-and-Forward Switching
  - Besteht aus N-Ports
- Port:
  - Latenz, Durchsatz

Datenfluss Model

- Datenflüsse als Flüsse von Paketen (Transfer Granularität)
- Datenfluss Model ist an Realität angelehnt garantiert aber optimalen Transfer (unter gegegebener Granularität)
- Server können Datenflüsse aktiv blockieren z.B. bei gefülltem I/O Cache
- Überholvorgänge innerhalb des Netzwerks sind nicht möglich
- Gegenwärtig nur eine Route

Paketübertragung

- Eine Netzwerkkomponente kann eine Menge von Paketen transferieren
  - Bis das Kabel mit Daten saturiert ist
  - Der weitere Datenfluss wird dann blockiert
- Bei Bearbeitung eines Pakets wird der Sender benachrichtigt
  - Diese darf nun weitere Pakete transferieren
  - Kaskade von nachrückenden Paketen kann aktiviert werden
- Es wird ein Fluss pro Ziel nach obigem Schema verwaltet
  - Engpässe die in Flussrichtung auftreten blockieren nicht den Transfer an andere Adressen
  - Anschaulich: ein Switch verwaltet pro Netzwerkadresse eine Warteschlange mit fester Länge

Software Model

- Anwendung
  - MPI Semantik
  - Nichtblockierende Operationen werden unterstützt
  - Rechenoperationen werden nur durch eine Dauer charakterisiert
  - Gleichmäßige Zuteilung der CPU Ressourcen zu Rechenjobs
- Mehrere (MPI-IO) Anwendungen können gleichzeitig gestartet werden
  - Insbesondere im I/O-Subsystem ist ein anderes Lastverhalten zu erwarten

Modellierung von MPI Befehlen

- Algorithmen sind austauschbar
  - Bei Ausführung Algorithmus spezifizieren
- Abarbeitung in endlichen Automaten
  - Programmierung der Zustände
  - Existierende Befehle können aufgerufen werden
  - Netzwerkoperationen
  - (Eigenverwaltete) Blockierung möglich
- Globale Sicht auf alle Prozesse
  - Metawissen soll Programmierung von einem Best-Case erlauben
  - Ermöglicht bspw. “Virtuelle” Bariere

Implementierung des Simulators

- Java (Version 5)
- GPL
- Sequenzieller Code
- Schreibt TAU-Trace oder HDTrace zur Analyse

Diskrete Ereignis-Simulation

Solange Ereignisse vorhanden sind:

- Bearbeite (ein) frühestes Ereignis durch delegation an zuständige Komponente
- Erzeugt ggf. weitere Ereignisse in der Zukunft

Job == Auftrag - durch blockierende Verarbeitung realisiert

- Start und Ende Ereignis
- Beispiel: Datenpakete werden durch Store-and-forward weitergeleitet
- Bearbeitung von Jobs wird von Komponenten selbstverwaltet (meist FIFO)

Validierung des Simulators

- Existierender Program getraced und simuliert
- Jacobi Verfahren - PDE: Iterative Lösung der Poisson Gleichung
  - 100 Iterationen
  - Master Prozess sammelt Ergebnisse ein
  - Keine I/O
- Cluster Model analog zu unserem 10 Knoten (Test-)Cluster

Jacobi - PDE

Prozessanzahl	Laufzeit in s	Simulierte Laufzeit	Simuliert/Real
1	47.30	47.35	1.001
2	24.79	24.93	1.006
3	17.3	17.54	1.014
4	13.54	13.75	1.016
5	11.56	11.82	1.022
6	10.09	10.33	1.024
7	9.16	9.44	1.030
8	8.39	8.73	1.041
9	8.00	8.26	1.033

Evaluation von I/O Optimierungen

Ziele

- Effizienz verschiedener Optimierungen soll überprüft werden
- Kollektive Optimierungen - Two-Phase* vs. serverseitige Optimierungen
- Entwicklung neuer I/O Optimierungsstrategien

Serverseitige Optimierungen

Stand der Technik

- I/O Zugriffe werden von Betriebsystem optimiert
  - Betriebsystem Cache und Write-Behind
- Typischerweise maximale Anzahl an Operationen die ans Betriebsystem gegeben werden
  - Cluster Dateisystem trifft Vorauswahl, wann - welche I/O-Operationen
- $\Rightarrow$ Die I/O Schicht verfügt nur über Teilwissen der I/O Anfragen
- Bei Lesezugriffen ist die Reihenfolge entscheident um Random-Access zu verhindern
- Bei Schreibzugriffen nicht so gravierend, da Write-Behind Optimierungen erlaubt

Server-Directed I/O

Theorie

- Cache-Optimierungsstrategie auf I/O Servern
- Datentransfer zwischen Client/Server wird durch Server bestimmt
  - Interessant für Nichtzusammenhängende Zugriffe
  - Geht über Kernel Optimierungen hinaus, da alle Anfragen berücksichtigt
- Der Server kennt seine eigenen I/O-Charakteristika am besten

Umsetzung im Simulator

- Cache-Schicht aggregiere Zugriffe (AggregationCache)
  - Nutze Wissen über Anfragen
- Erweiterung sortiert auch die Zugriffe um (Server-Directed)
  - Nutze Wissen über I/O-Subsystem Charakteristika
- Im Moment wird Datentransfer nur bei lesenden Zugriffen angepasst
  - Write-Behind erlaubt schon sehr gute Aggregationen
  - Optional kann man Data-Sieving noch in Cache Schicht einbauen

Evaluation

Simuliertes Cluster

- 10 Clients, Zugriffsmuster Simple-Strided [1,2,...,10,1,2,...,10]
- 10 Server
  - 1000 MB Cache (RAM)
  - Platte - 50 MB/s, 10 ms mittlere Zugriffszeit, track-to-track 1 ms
    - bis 5 MByte track-to-track seek sonst mittlerer Zugriff
- Sternförmige Vernetzung
  - Netzwerkkarten – Durchsatz 100 MB/s, Latenz 0.2 ms
  - Switch Durchsatz limitiert auf 1000 MB/s
- Datenverteilung: Stripping mit 64 KByte

Experimente

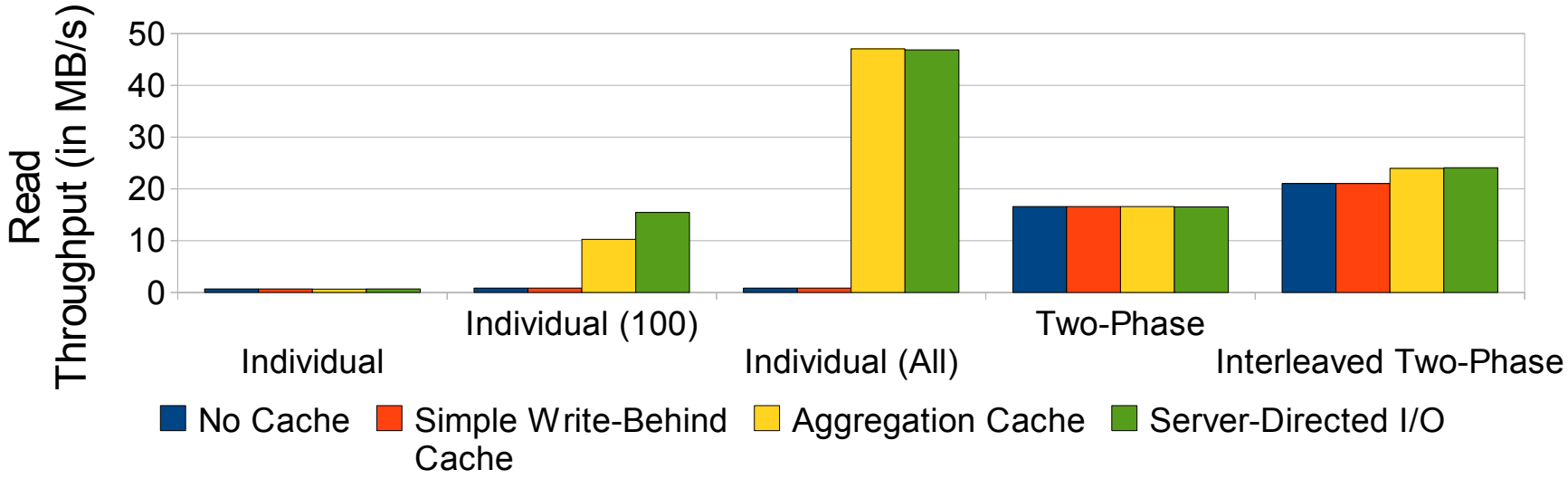
Sequenzielle E/A einer Datei mit 1000 MByte

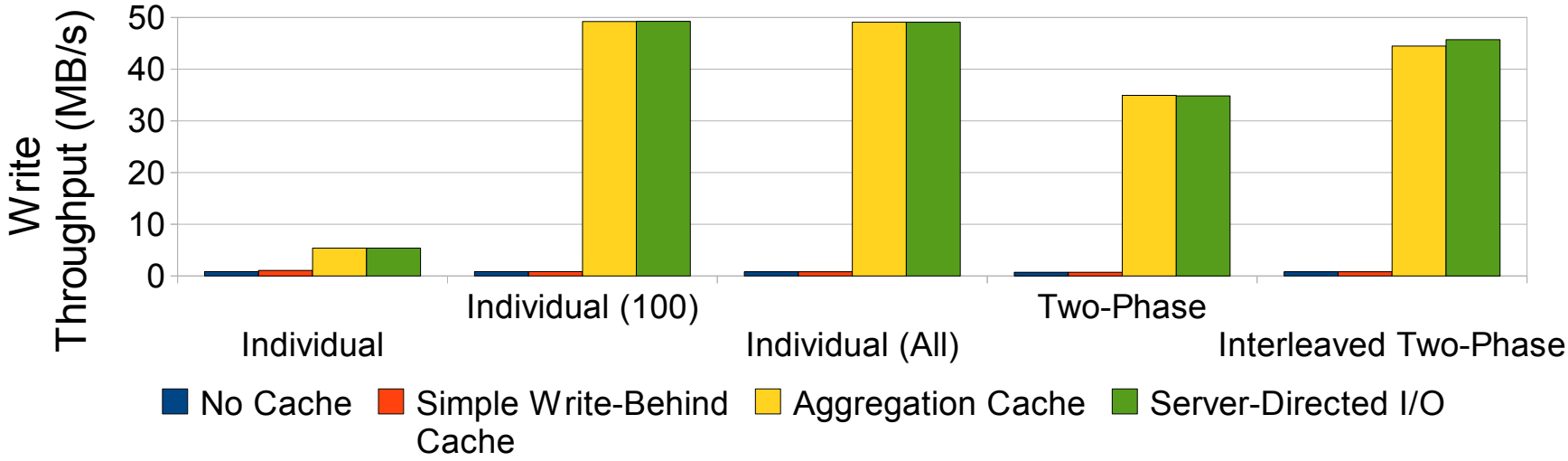
- Unabhängige I/O mit einem, 100 oder alle Blöcke
- Two-Phase
- Interleaved Two-Phase

Erwartungen

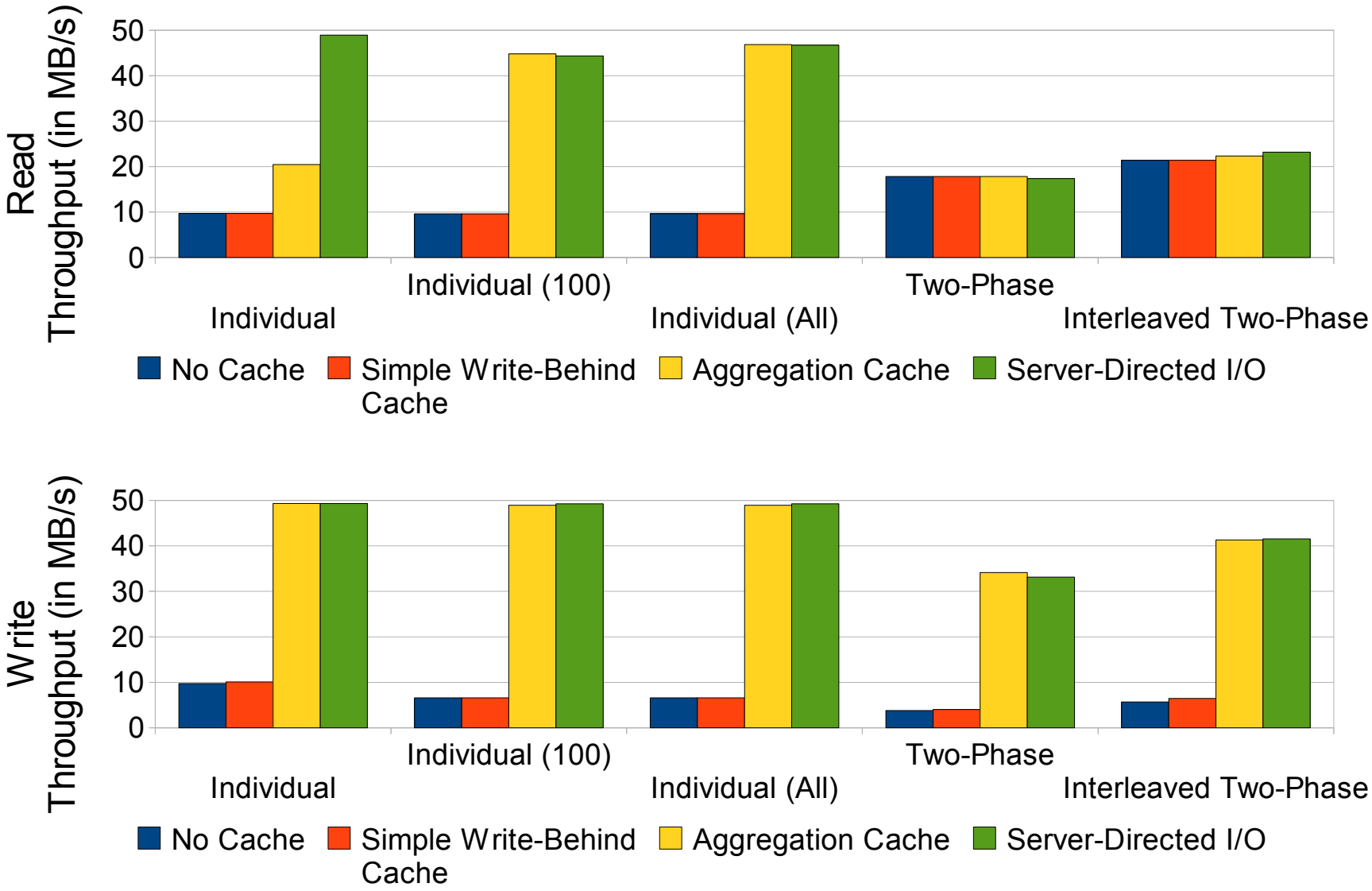
- Flaschenhals sind die Festplatten
- Maximal 50 MB/s pro Client und Server
- Two-Phase weniger als 50 MB/s wegen Netzwerktransport

Vergleich von Optimierungsstrategien, Blockgröße: 5 KB





Vergleich von Optimierungsstrategien, Blockgröße: 512 KB

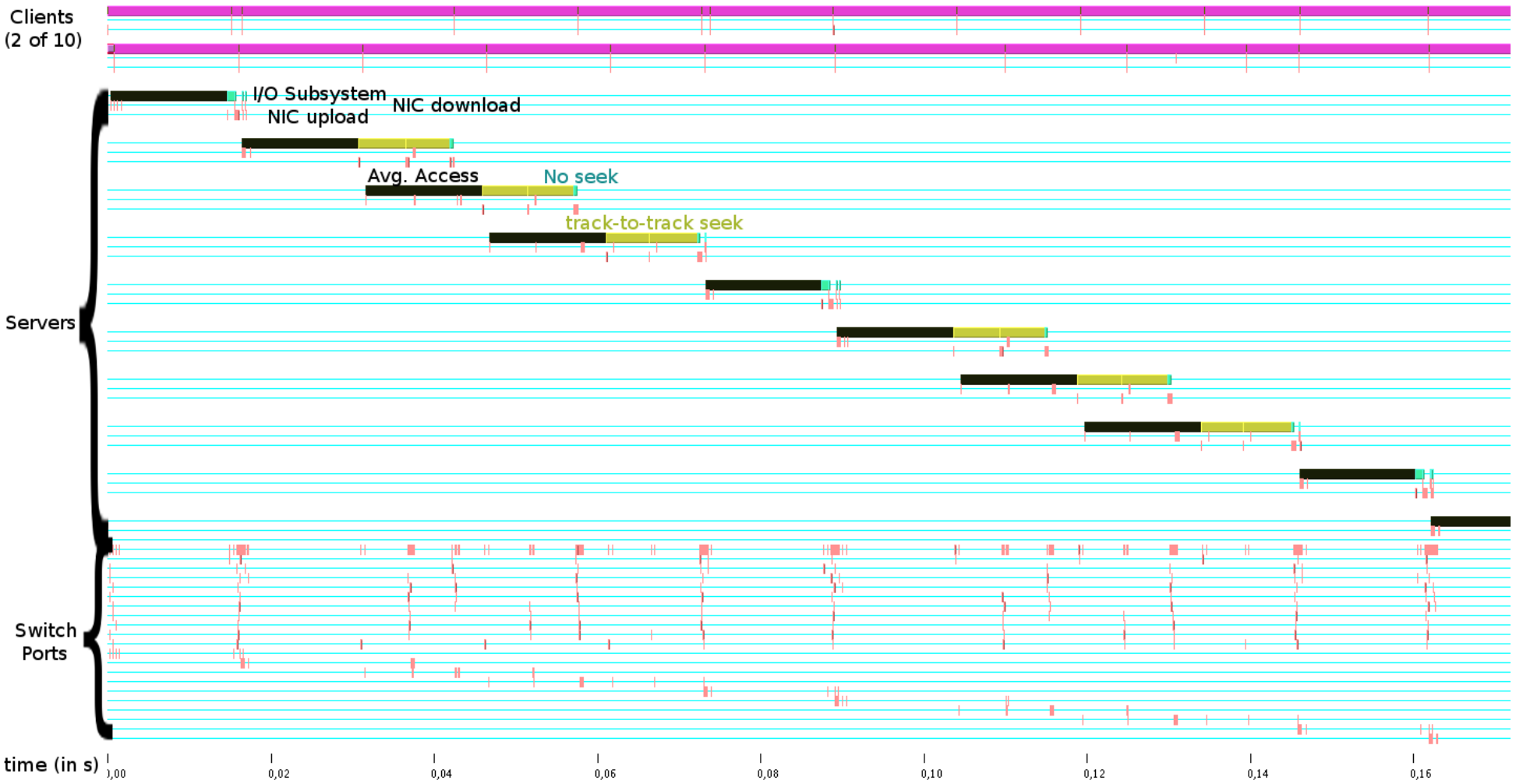


Warum ist ServerDirected bei lesendem Zugriff teilweise so langsam?

Analyse

- Problemfall: Blockgröße 5 KByte
- Bei individueller I/O nur 1 MB/s
- Beim nicht zusammenhängendem Zugriff mit 100 Blöcken nur 15 MB/s
- Mit Viewer Simulationsergebnisse betrachten
- Im folgenden gekürzte (und beschriftete) Screenshots
  - Nur ein Teil der Server und Clients gezeigt

Unabhängige zusammenhängende I/O - 5 KB - Startphase



Unabhängige zusammenhängende I/O - 5 KB - Später

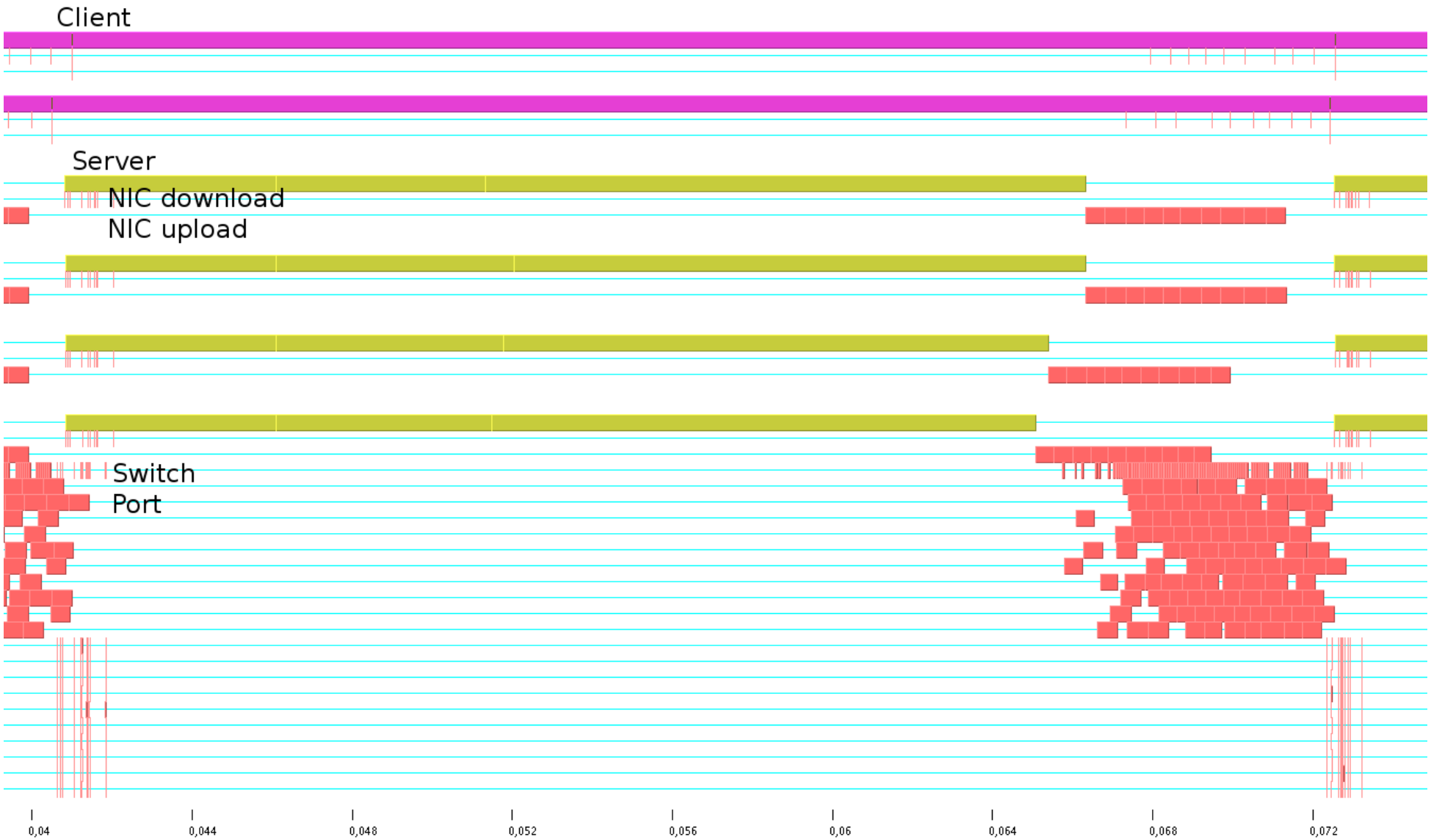
These

I/O Anfragen der 10 Clients sind später über die 10 Server verteilt
$\Rightarrow$ Keine Aggregation auf den Servern möglich

Prüfung

- Test mit mehr Clients – 100 Clients
- NoCache = 0.9 MB/s - (nahezu Random Zugriffe)
- ServerDirectedIO = 4.4 MB/s

Eine Iteration - 5 KB - Unabhängige 100 Blöcke/Iteration



Analyse

- Pro Client 100 * 5 KB = 500 KB
- Durch Zugriffsmuster Clients brauchen Blöcke von jedem Server
- Aggregation bündelt Anfragen sinnvoll zusammen
- Sobald I/O beendet ist werden alle Clients aktiv
  - $\Rightarrow$ implizite Synchronisation
- Leerzeiten auf Server

Fazit

- Einfaches Model für Simulation von MPI und Cluster Hardware
- Geeignet für Simulation von heterogenen Umgebungen
- Alternative Implementierungen für MPI Befehle evaluieren
- Ergebnisse am Beispiel von I/O Optimierungen
  - Server-Seitige Optimierung in den vielen Fällen ausreichend
- Visualisierung des Systemverhaltens vereinfacht Analyse