Parallele verteilte Dateisysteme

Hochleistungs-Ein-/Ausgabe

Michael Kuhn

Wissenschaftliches Rechnen
Fachbereich Informatik
Universität Hamburg

2017-05-05

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

1 Parallele verteilte Dateisysteme
- Orientierung
- Konzepte
- Leistungsüberlegungen
- Lustre
- OrangeFS
- Leistungsanalyse
- Ausblick und Zusammenfassung

2 Quellen

E/A-Schichten

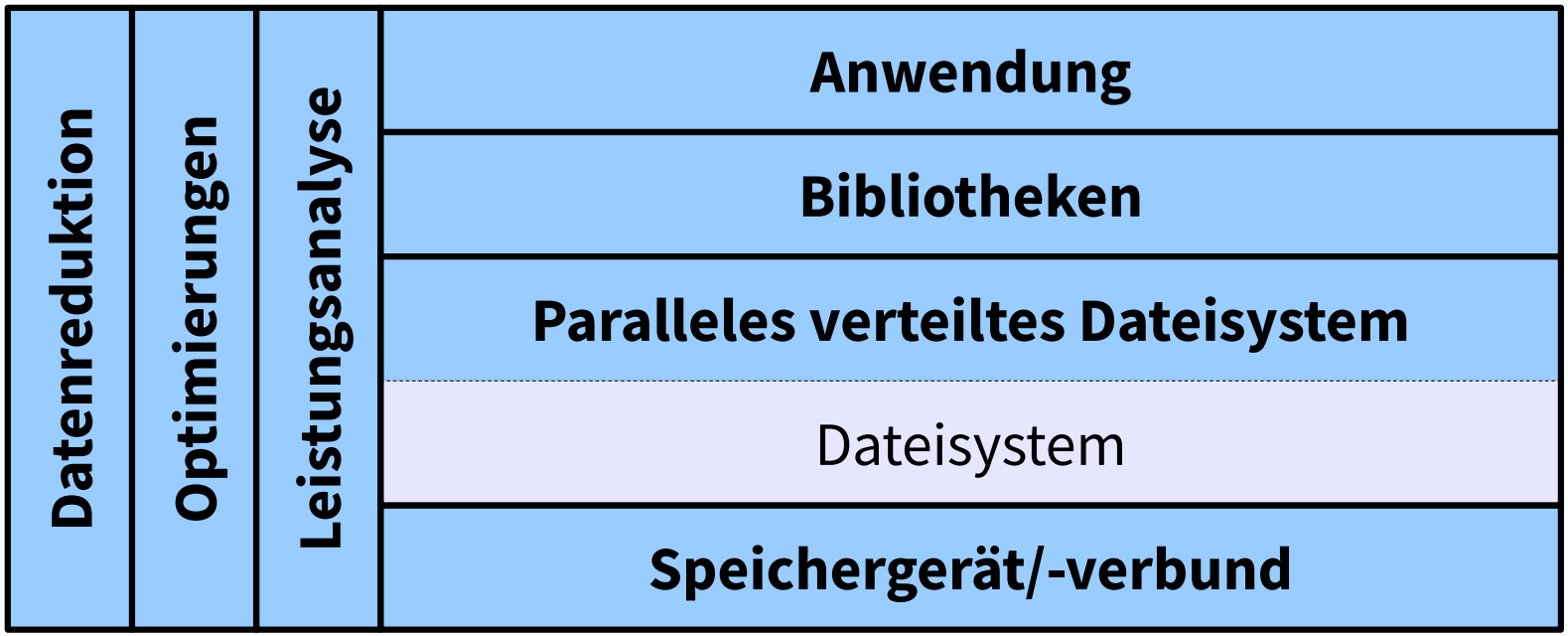


Abbildung: E/A-Schichten

Definition

- Parallele Dateisysteme
  - Erlauben parallelen Zugriff auf gemeinsame Ressourcen
  - Zugriff soll möglichst effizient erfolgen können
- Verteilte Dateisysteme
  - Daten und Metadaten sind über mehrere Server verteilt
  - Einzelne Server haben keine vollständige Sicht
- Benennung uneinheitlich
  - Oft nur „paralleles Dateisystem“ oder „Clusterdateisystem“

Definition…

- Im Hochleistungsrechnen werden üblicherweise immer parallele verteilte Dateisysteme eingesetzt
  - NFS nur für nicht leistungskritische Anwendungen
- Lokale Dateisysteme erlauben auch parallelen Zugriff
  - Sperren z.B. via `flock` oder `lockf`
- Verteilung erfordert Zusatzaufwand
  - Algorithmen für verteilte Sperren etc.

- Home-Verzeichnisse können über NFS realisiert werden
  - Werden der Einfachheit halber aber häufig auch durch das parallele verteilte Dateisystem bereitgestellt
- Paralleler Zugriff ist in lokalen Dateisystem relativ einfach realisierbar
  - Alle Zugriffe werden durch das VFS koordiniert

Definition…

- Storage Area Network (SAN)
  - Stellt nur Blockgeräte über das Netzwerk bereit
  - Darauf kann beliebiges Dateisystem verwendet werden
  - Parallele verteile Dateisysteme können SANs nutzen
- Network Attached Storage (NAS)
  - Abstrahiert von Speichergeräten
  - Stellt direkt ein Dateisystem bereit
    - Üblicherweise NFS oder SMB

Grundlagen

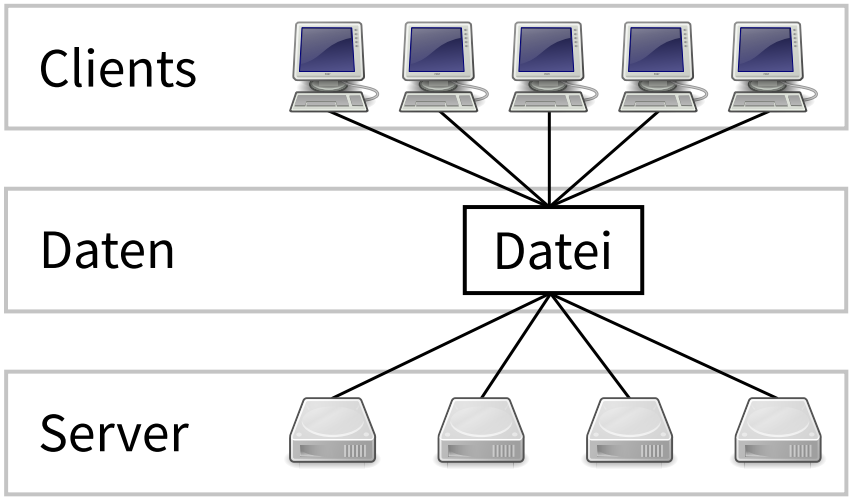


Abbildung: Paralleler Zugriff und Datenverteilung

- Clients greifen parallel auf eine gemeinsame Datei zu
- Datei ist über mehrere Server und Speichergeräte verteilt
  - Hoher Durchsatz und hohe Kapazität

Architektur

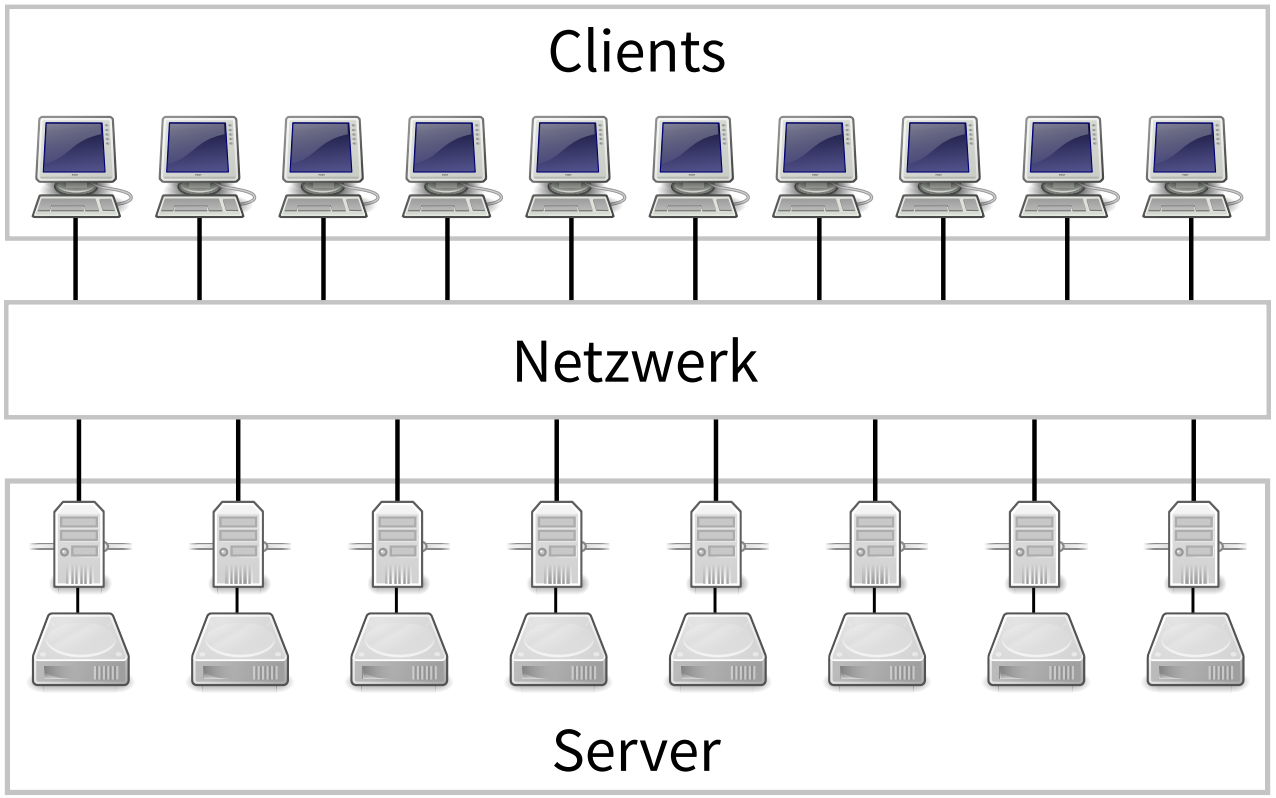


Abbildung: Paralleles verteiltes Dateisystem

Architektur…

- Trennung in Clients und Server
  - Keine gegenseitige Beeinflussung
- Clients führen parallele Anwendungen aus
  - Haben über das Netzwerk Zugriff auf Dateisystem
  - Üblicherweise kein lokaler Speicher und kein direkter Zugriff auf Speichergeräte der Server
- Daten- und Metadatenserver
  - „Einfache“ Controller oder vollwertige Server
  - Unterschiedliche Server für Daten und Metadaten
    - Produzieren unterschiedliche Zugriffsmuster

Architektur…

- Clients müssen mit Servern kommunizieren
- Unterschiedliche Kommunikationsansätze
  - Clients kennen zuständigen Server (verbreiteter)
  - Clients kontaktieren einen zufälligen Server
- Unterschiedliche Serververhalten
  - Server teilen Clients zuständigen Server mit
  - Server leiten Anfrage transparent weiter
- In beiden Fällen Vor- und Nachteile

Architektur…

- Clients kennen zuständigen Server
  - Vorteile: Einfaches Kommunikationsprotokoll (eins zu eins), keine Kommunikation zwischen Servern
  - Nachteile: Verteilungslogik muss von Clients implementiert werden, zusätzliche clientseitige Informationen notwendig
- Clients kontaktieren einen zufälligen Server
  - Vorteile: Clients müssen Daten-/Metadatenverteilung nicht kennen, Last-Balancierung einfacher umzusetzen
  - Nachteile: Zugriffe erfordern zusätzliche Nachrichten, komplexeres Kommunikationsprotokoll

Schnittstelle

- Zugriff auf das Dateisystem über E/A-Schnittstelle
  - Üblicherweise standardisiert
  - Proprietäre für mehr Funktionen und Leistung
- Schnittstelle besteht aus Syntax und Semantik
  - Syntax legt verfügbare Operation fest
  - Semantik legt Verhalten der Operationen fest
- Häufig POSIX-Schnittstelle
  - Standardisiert und portabel

Schnittstelle…

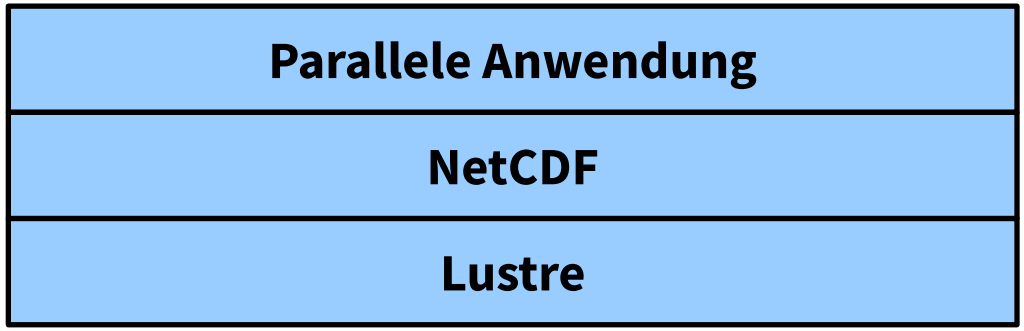


Abbildung: E/A-Stack aus Anwendungssicht

- Anwendungen nutzen höher abstrahierende Schnittstellen
  - NetCDF bietet selbst-beschreibendes Datenformat
- Paralleles verteiltes Dateisystem kümmert sich um effizienten Zugriff und Verteiltung

Semantik

- POSIX hat strenge Konsistenzanforderungen
  - Änderungen müssen nach `write` global sichtbar sein
  - E/A soll atomar geschehen
- POSIX für lokale Dateisysteme
  - Anforderungen dort einfach zu unterstützen
  - Alles über das VFS abgewickelt
- Kleinigkeiten änderbar
  - `strictatime`, `relatime` und `noatime` für Verhalten bezüglich Zugriffszeitstempel
  - `posix_fadvise` für Ankündigung des Zugriffsmusters

Semantik…

- Gegensatz: Network File System (NFS)
  - Selbe Syntax, deutlich unterschiedliche Semantik
- Sogenannte Sitzungssemantik
  - Änderung für andere Clients nicht sichtbar
    - Nur innerhalb der Sitzung
  - Für andere Clients erst nach deren Ende
  - `close` schreibt Änderungen und liefert eventuelle Fehler
- Später: MPI-IO
  - Weniger strikt für höhere Skalierbarkeit

Realität

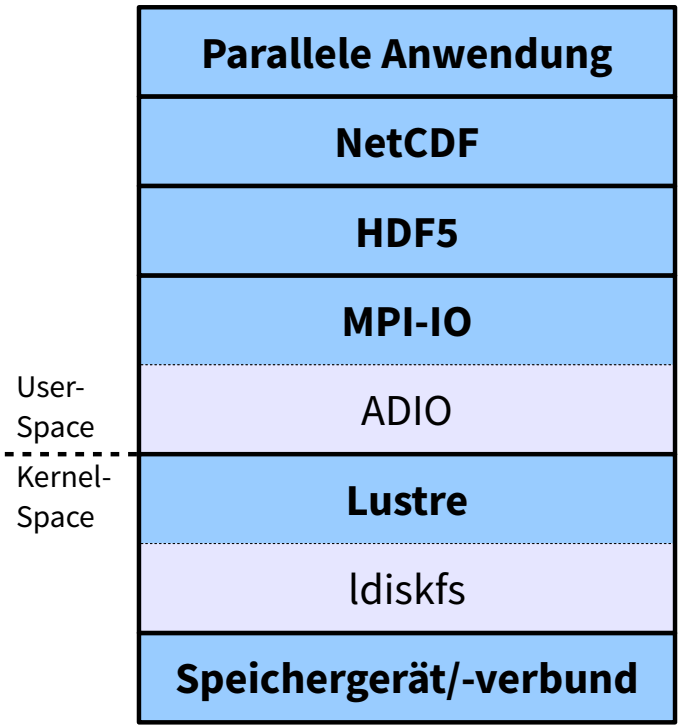


Abbildung: E/A-Stack im HPC

- Datenumwandlung
  - Transport durch alle Schichten
  - Verlust von Information
- Komplexes Zusammenspiel
  - Optimierungen und Workarounds pro Schicht
  - Informationen über andere Schichten
- Komfort vs. Leistung
  - Strukturierte Daten in Anwendung
  - Bytestrom in POSIX

Realität…

Abstraktion	Schnittstelle	Datentypen	Kontrolle
Hoch ⇕	NetCDF	Strukturen	Grobgranular
	MPI-IO	Elemente	
Niedrig ⇕	POSIX	Bytes	Feingranular

Abbildung: Abstraktionsstufen

- Hohe Abstraktion bietet viel Komfort aber wenig Kontrolle
  - „Schreibe Matrix `m`“
- Niedrige Abstraktion erlaubt genaues Tuning
  - „Schreibe n Bytes an Offset `m`“

Beispiele

- GPFS (IBM)
- Lustre
- OrangeFS (früher PVFS)
- CephFS (Red Hat)
- BeeGFS (Fraunhofer, früher FhGFS)
- GlusterFS (Red Hat)

Generelles

- E/A ist im Vergleich zu Berechnung teuer
  - Kontextwechsel, langsame Speichergeräte etc.
- Parallele verteilte E/A muss über Netzwerk abgewickelt werden
  - Einschränkungen bezüglich Durchsatz und Latenz
- Neuartige Konzepte wie Burst-Buffer
  - Nehmen Daten temporär auf
  - Gleichförmiges Weiterreichen an Dateisystem
  - Z. B. knoten-lokale nichtflüchtige Speicher (NVRAM)

Daten

- Daten werden potentiell von mehreren Clients zugegriffen
  - Überlappend vs. nicht überlappend
  - Lesezugriffe meist unproblematisch
- Überlappende Schreibzugriffe
  - Stark abhängig von der E/A-Semantik
  - Benötigt üblicherweise Sperren
  - Daher häufig verteilte Sperrenverwaltung
- Nicht überlappende Schreibzugriffe
  - Potentielle Leistungsprobleme durch grobgranulare Sperren

Daten…

- Datenverteilung relevant für Leistung
- Realisiert über Verteilungsfunktionen
  - Üblicherweise simples Round-Robin
  - Manchmal durch Benutzer steuerbar
  - Z. B. auch Unterstützung heterogener Zugriffsmuster
- Anzahl der zu kontaktierenden Datenserver
  - Nicht zu wenige, da sonst Durchsatz beschränkt
  - Nicht zu viele, da sonst zu viel Overhead

Daten…

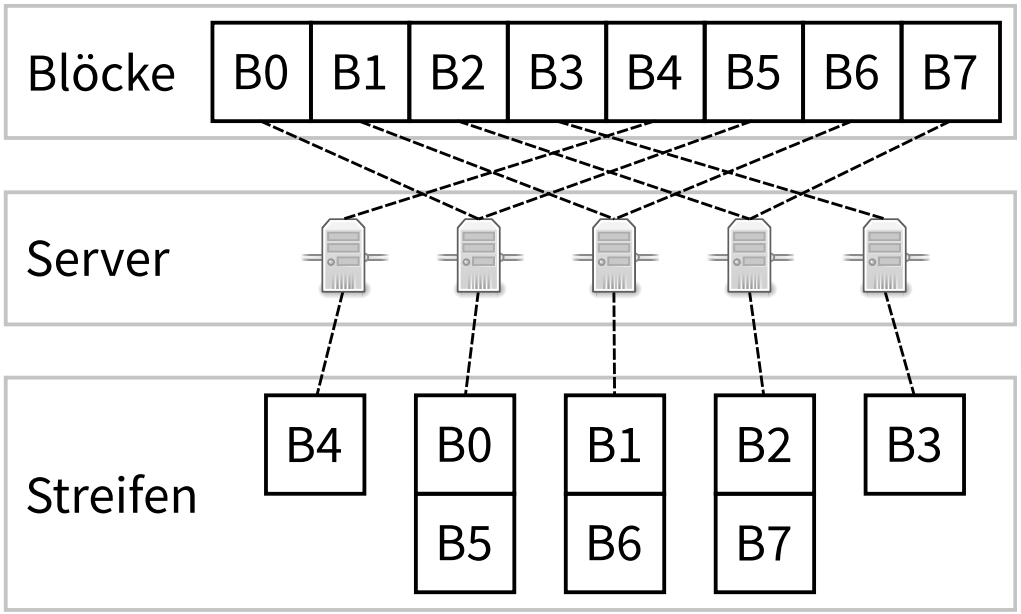


Abbildung: Round-Robin-Datenverteilung

- Datei aus Blöcken, wird in Streifen auf Server verteilt
  - In diesem Fall entspricht Blockgröße der Streifengröße
- Verteiltung muss nicht auf dem ersten Server starten

Metadaten

- Metadaten werden von mehreren Clients zugegriffen
  - Lesezugriffe üblicherweise wieder unproblematisch
  - Parallele Änderungen an Größe, Zeitstempel etc.
- Metadaten für eine Datei von einem Server verwaltet
  - Aktualisierungen daher meist auch seriell
- Potentiell mehrere Millionen Clients
  - DDoS durch Clients auf Metadatenserver

Metadaten…

- Verteilte Ansätze zur Metadatenverwaltung
  - Veränderliche Metadaten nicht zentral speichern
    - Z. B. Größe und Zeitstempel
  - Berechnung zur Laufzeit
    - Client kontaktiert alle relevanten Datenserver
    - Aktualisierung auf Kosten der Abfrage beschleunigt
- Metadatenverteilung analog zu Datenverteilung
  - Bestimmung des Servers z. B. durch Hashing des Pfades
  - Muss üblicherweise deterministisch sein
    - Clients müssen autonom zuständige Server ermitteln können
  - Ein Objekt üblicherweise von einem Server verwaltet
    - Neuerdings Ausnahmen bei Verzeichnissen

Metadaten…

- Viele Metadatenoperationen sind inhärent seriell
  - Z. B. Pfadauflösung

1. `/foo/bar`
   1. Inode des Wurzelverzeichnisses lesen
   2. Zugriffsrechte überprüfen
   3. Wurzelverzeichnis lesen und nach `foo` durchsuchen
2. `/foo/bar`
   1. Inode des Verzeichnisses lesen
   2. Zugriffsrechte überprüfen
   3. Verzeichnis lesen und nach `bar` durchsuchen
3. `/foo/bar`
   1. Inode der Datei lesen
   2. Zugriffsrechte überprüfen
   3. Auf Datei zugreifen

Metadaten…

Technologie	Gerät	IOPS
HDD	7.200 RPM	75–100
	10.000 RPM	125–150
	15.000 RPM	175–210
SSD	Intel X25-M G2	8.600
	OCZ Vertex 4	85.000–90.000
	Fusion-io ioDrive Octal	1.000.000+

Tabelle: IOPS für ausgewählte HDDs und SSDs [4]

- Getrennte Server erlauben gezielte Optimierung
  - Z. B. Festplatten für Daten, Flashspeicher für Metadaten
  - Unterschiedliche Preise (Faktor ≥ 10)
  - Metadaten machen ca. 5 % der Gesamtdaten aus
- Software muss Leistung auch ausnutzen können

Leistungsgrößen

- Parallele verteilte Dateisysteme erlauben riesige und hochperformante Speichersysteme
- Blizzard (DKRZ, GPFS)
  - Größe: 7 PB
  - Durchsatz: 30 GB/s
- Mistral (DKRZ, Lustre)
  - Größe: 54 PiB
  - Durchsatz: 450 GB/s (5,9 GB/s pro Knoten)
  - IOPS: 80.000 Operationen/s (nur Phase 1)
- Titan (ORNL, Lustre)
  - Größe: 40 PB
  - Durchsatz: 1,4 TB/s

Übersicht

- Eines der bekanntesten parallelen verteilten Dateisystemen
- Open Source (GPLv2)
  - $>$ 550.000 Zeilen Code
- Unterstützung für Linux
  - Name abgeleitet von Linux und Cluster
- Sehr weit verbreitet
  - Mehr als die Hälfte der TOP100
  - Mehr als ein Drittel der TOP500

Geschichte

- 1999: Entwicklungsbeginn
  - Forschungsprojekt an der Carnegie Mellon University, geleitet von Peter Braam
- 2001: Gründung Cluster File Systems
- 2007: Kauf durch Sun
  - Integration in HPC-Hardware
  - Kombination mit ZFS
- 2010: Kauf durch Oracle
  - Einstellung der Entwicklung
- Weiterentwicklung durch Community
  - Intel (ehemals Whamcloud), Seagate (ehemals Xyratex), OpenSFS, EOFS etc.

Geschichte… [2]

- Version 2.3 (Oktober 2012)
  - Experimentelle Unterstützung für ZFS
- Version 2.4 (Mai 2013)
  - Distributed Namespace (DNE)
  - ZFS für Daten und Metadaten
- Version 2.5 (Oktober 2013)
  - Hierarchical Storage Management (HSM)
- Version 2.6 (Juli 2014)
  - Experimentelle Unterstützung für verteilte Verzeichnisse
- Version 2.7 (März 2015)
  - Bessere Unterstützung für verteilte Verzeichnisse (experimentell)
  - Objektplatzierung via `lfs setstripe`

Geschichte… [2]

- Version 2.8 (März 2016)
  - Finale Unterstützung für verteilte Verzeichnisse
  - Metadatenoperationen über mehrere Metadatenserver hinweg
- Version 2.9 (Dezember 2016)
  - Kerberos-Unterstützung (Authentifizierung und Verschlüsselung)
  - Unterverzeichnisse können gemountet werden
  - Unterstützung für 16 MiB große RPCs
  - Unterstützung für `ladvise`
- Version 2.10 (Juni 2017)
  - Dateisystemschnappschüsse
  - Projekt-Quotas
  - Progressive Datei-Layouts

Geschichte…

```
1 $ lfs mkdir --index 0 /lustre/home
2 $ lfs mkdir --index 1 /lustre/scratch
3 $ lfs mkdir --count 3 /lustre/striped
```

Listing 1: DNE in Lustre

- DNE erlaubt unterschiedliche Verzeichnisse auf unterschiedliche Metadatenserver zu verteilen
  - `/scratch` für große Dateien
  - `/home` üblicherweise mit vielen kleineren
  - Statischer Ansatz, manuelle Konfiguration
- Unterstützung für verteilte Verzeichnisse
  - `/striped` über drei Metadatenserver verteilt

Architektur

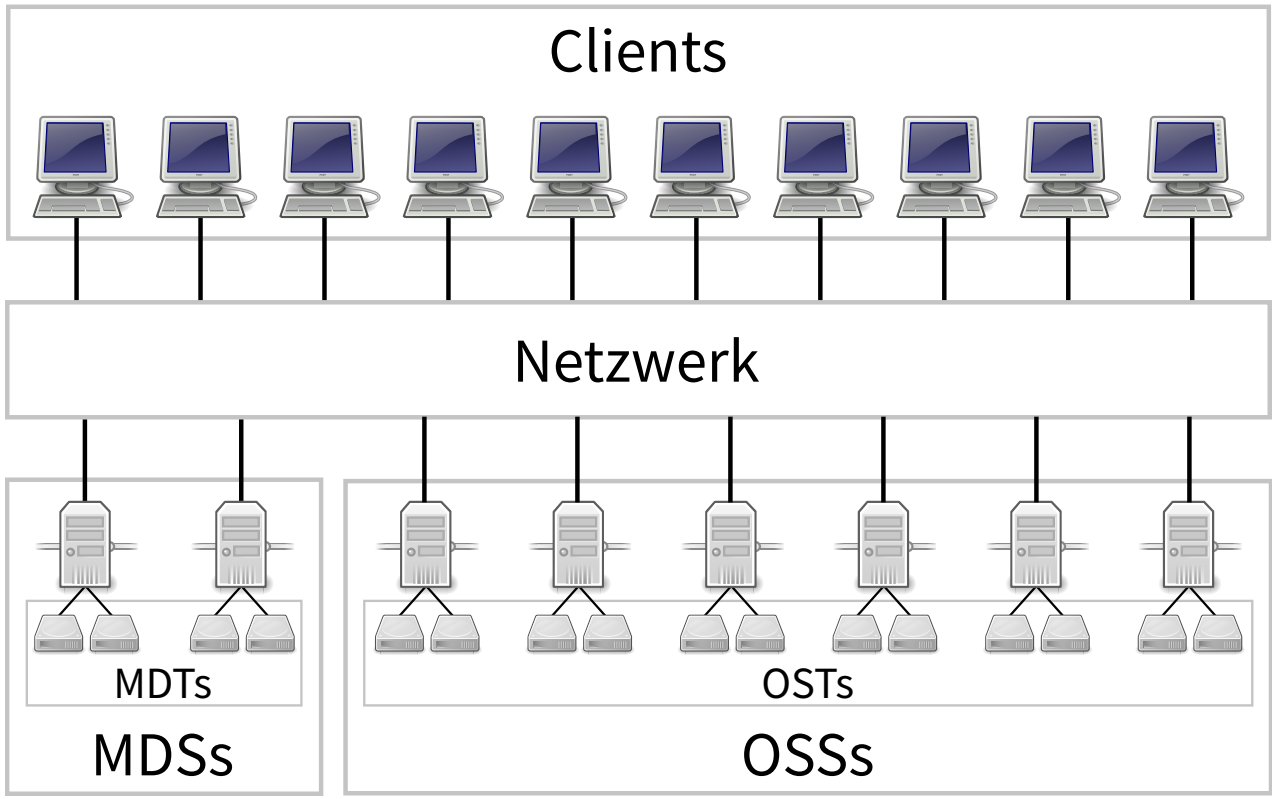


Abbildung: Lustre-Architektur

Architektur…

- Object Storage Servers (OSSs)
  - Verwalten Daten
  - Objekt-basierter Zugriff auf Byte-Ebene
  - Speicherung auf Object Storage Targets (OSTs)
- Metadata Servers (MDSs)
  - Verwalten Metadaten
  - Nicht in eigentliche E/A involviert
  - Speicherung auf Metadata Targets (MDTs)
- Verteilung über Targets, nicht über Server
  - Ein Server kann mehrere Targets verwalten

Architektur…

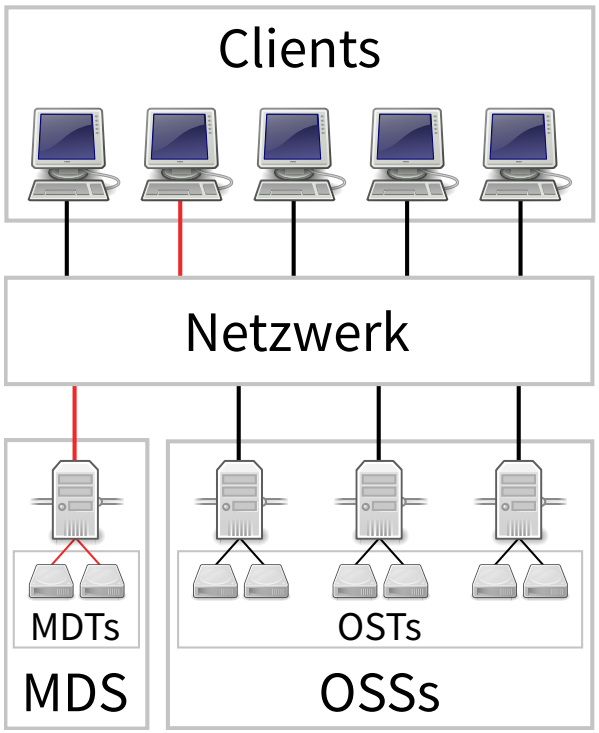
- Daten- und Metadatenserver nutzen lokales Dateisystem
  - Üblicherweise ldiskfs (ext4-Fork)
  - Alternativ ZFS (Data Management Unit)
    - Dadurch kein POSIX-Overhead
- Kein direkter Zugriff auf Speichergeräte durch Clients
  - Clients senden Anfragen an Server
  - Server führen Operationen durch
  - Server schicken Ergebnisse an Clients

Architektur…

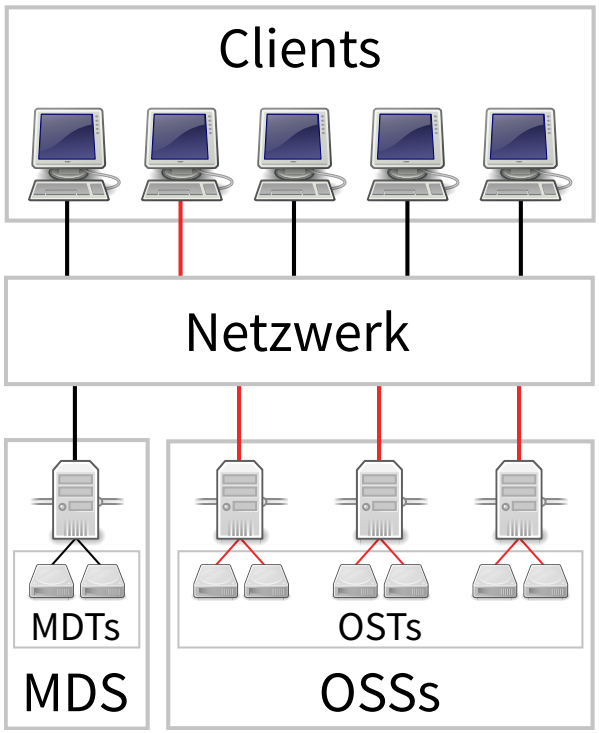
- Lustre nutzt ein direktes Kommunikationsprotokoll
  - Clients kennen zuständigen Server
- Server müssen sich nicht gegenseitig kennen
  - Server müssen Managementserver kennen
  - Metadatenserver müssen sich u. U. untereinander kennen

1. Clients kontaktieren Managementserver
   - Konfiguriert über `/etc/fstab` bzw. Mountoption
   - Liefert weitere Informationen über vorhandene Server etc.
2. Clients kontaktieren zuständigen Metadatenserver
   - Dieser liefert Informationen über zuständige Datenserver
3. Clients kontaktieren zuständige Datenserver

Architektur…



(a) Metadatenzugriff



(b) Datenzugriff

(a) Metadatenserverzugriff nur für initiales Öffnen
(b) Danach direkter paralleler Zugriff auf Datenserver

Unterstützung

- Lustre ist ein Kernel-Dateisystem
  - Sowohl Client als auch Server
- Client unterstützt (relativ) aktuelle Kernel
  - Seit 3.12 direkt in den Kernel integriert (aber veraltet)
  - Unterstützung neuer Kernel dauert manchmal eine Weile
- Server unterstützt nur ausgewählte Enterprise-Kernel
  - Z. B. Red Hat Enterprise Linux (oder CentOS)
  - Hauptsächlich verursacht durch ldiskfs
  - Mit ZFS auch Unterstützung des Standard-Kernels

Funktionalität

- Verteilte Sperrenverwaltung
  - Sowohl für Daten als auch Metadaten
  - Überlappende Lesesperren und nicht-überlappende Schreibsperren auf Byte-Ebene
  - Sperren können mit Mountoption `nolock` deaktiviert werden
- Unterstützung für explizite Sperren
  - Mountoptionen `noflock`, `localflock` und `flock`
- POSIX-konform
  - POSIX-Schnittstelle durch VFS
  - Keine native Unterstützung für MPI-IO

Funktionalität…

- Hierarchical Storage Management
  - Wichtige Anforderung für große Speichersysteme
  - Unterstützt mehrere „Tiers“
    - Festplatten, Tapes etc.
  - Metadaten werden weiterhin durch Lustre verwaltet
  - Daten werden transparent in andere Tiers verschoben
- Hochverfügbarkeit
  - Unterstützung von Failover-Mechanismen
  - Aktiv/Passiv- und Aktiv/Aktiv-Konfigurationen

Funktionalität…

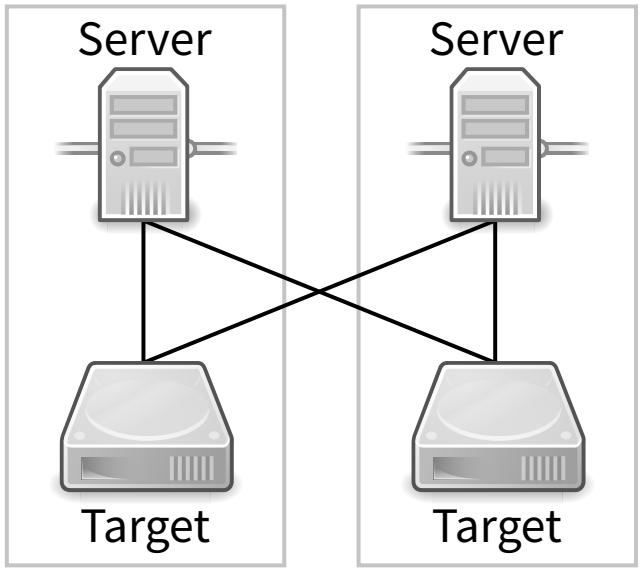


Abbildung: Aktiv/Aktiv-Failover-Konfiguration

- Server können jeweils Rolle des anderen übernehmen
- Kann für unterbrechungsfreie Upgrades genutzt werden

Überblick

- Open Source (LGPL)
  - $>$ 250.000 Zeilen Code
- Entwicklung durch Clemson University, Argonne National Laboratory und Omnibond
- Weiterentwicklung von PVFS
  - 2007: Start als Entwicklungszweig
  - 2010: Ersetzt PVFS als Hauptversion

Funktionalität [3]

- Verteilte Metadaten und Verzeichnisse
  - Verteilte Verzeichnisse seit Version 2.9
- Läuft komplett im User-Space
- Sehr gute MPI-IO-Unterstützung
  - Natives Backend in ROMIO
- Unterstützung für POSIX-Schnittstelle
  - FUSE-Dateisystem
  - Optionales Kernelmodul

Funktionalität…

- OrangeFS ist nicht POSIX-konform
  - Garantiert atomare Ausführung nicht-zusammenhängender und nicht-überlappender Zugriffe
  - Bezieht sich auf Lese- und Schreiboperationen
  - Unterstützt somit (nicht-atomares) MPI-IO
- Ausreichend für viele Anwendungsfälle
  - Striktere Semantik allerdings nicht unterstützt
  - MPI-IO-Atomic-Modus mangels Sperren nicht verfügbar

Lustre

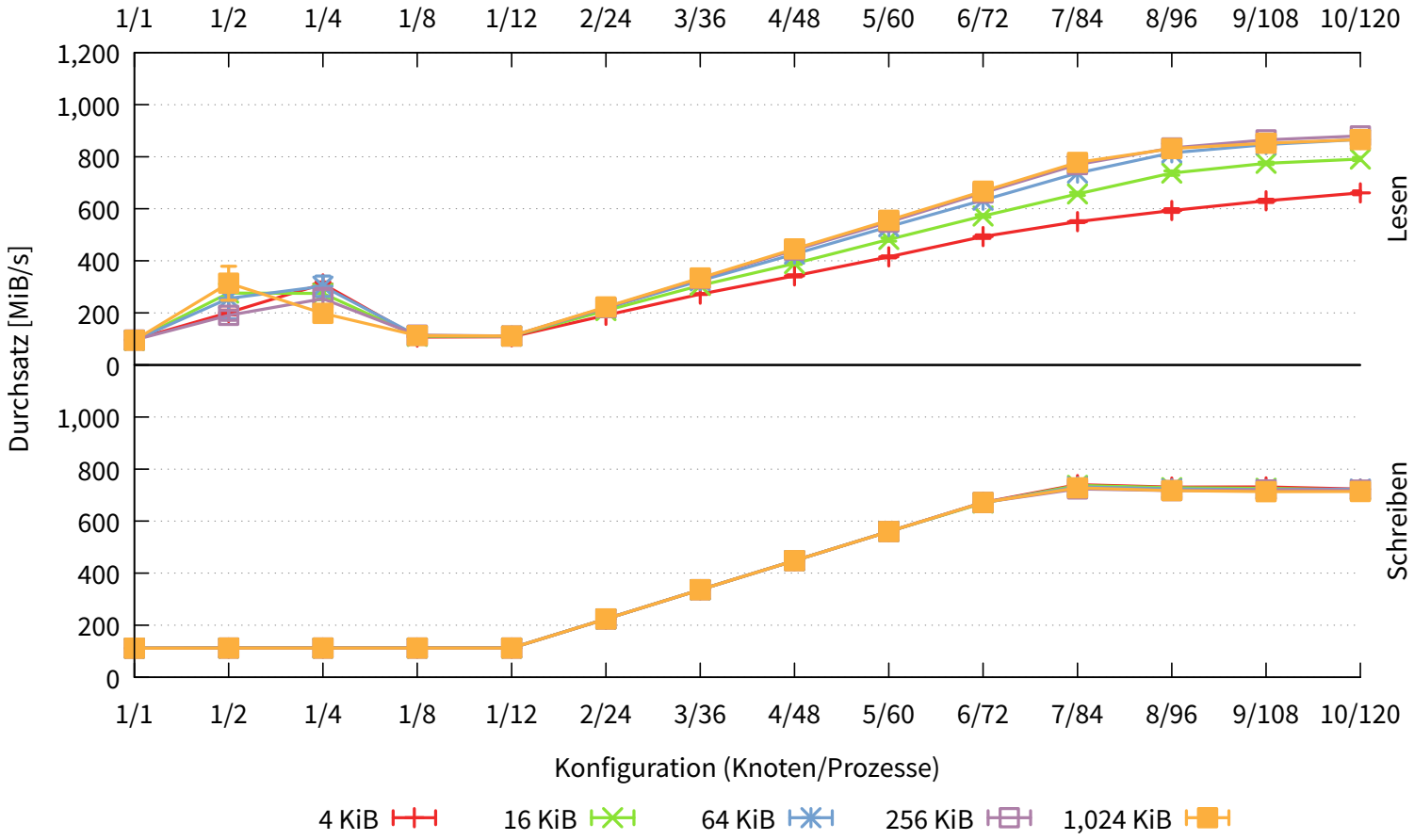


Abbildung: Lustre mit prozess-lokalen Dateien

Lustre…

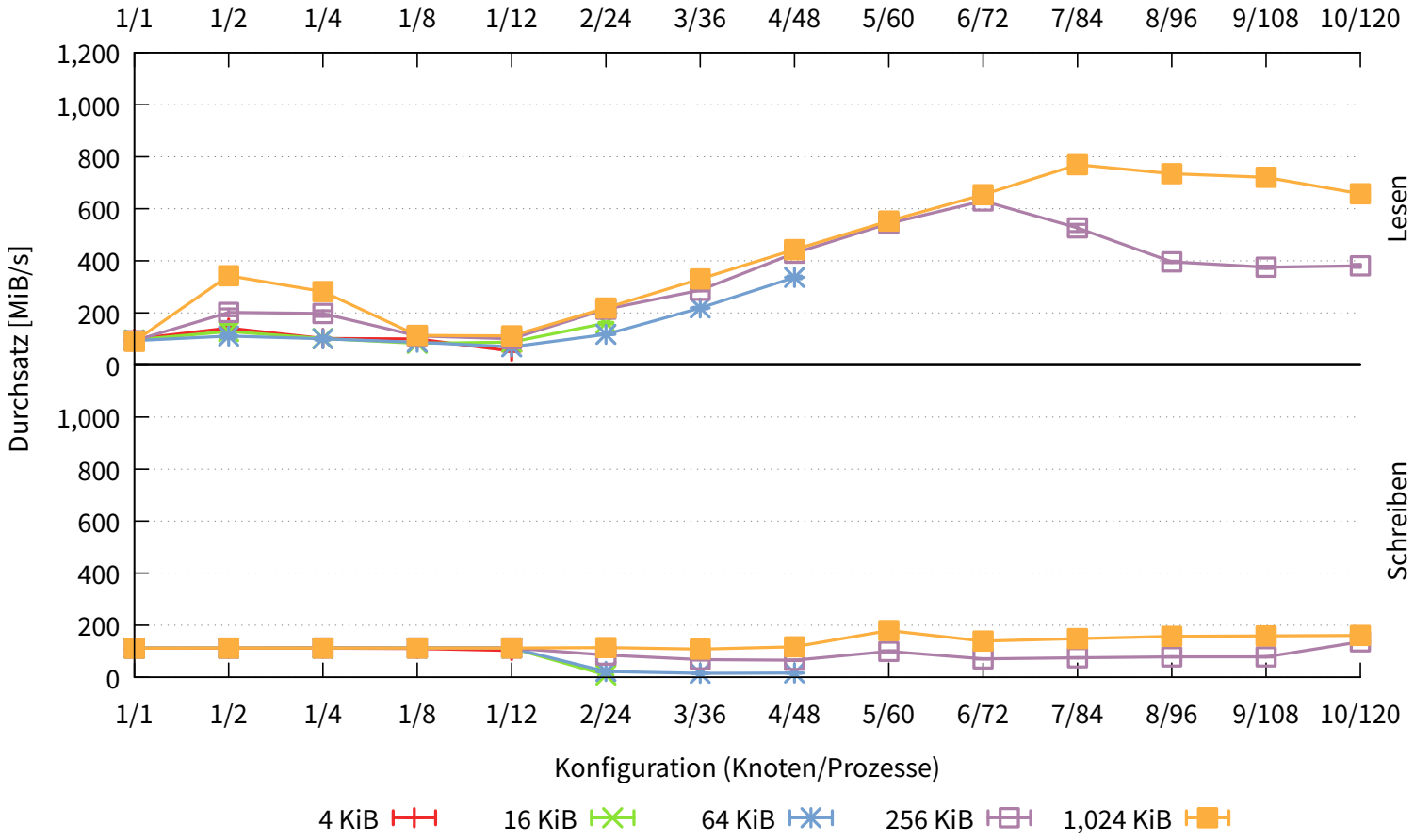


Abbildung: Lustre mit gemeinsamer Datei

Lustre…

- Lustre hochoptimiert für viele parallele Clients
  - Kein Leistungseinbruch mit wachsender Clientzahl
- Allerdings nur für prozess-lokale Dateien
  - Gemeinsame Dateien skalieren nicht
  - Auswirkung der POSIX-Einschränkungen
  - Hohe Leistung möglich aber kompliziert
- Diverse Optimierungen
  - Aggregiert Schreibzugriffe im Hauptspeicher
  - Führt Readahead durch

- Für hohe Leistung mit gemeinsamen Dateien ist es notwendig die Zugriffe an den Streifengrenzen auszurichten
  - Außerdem möglichst 1:1-Zugriffe, d. h. ein Client kommuniziert mit nur einem Server

OrangeFS

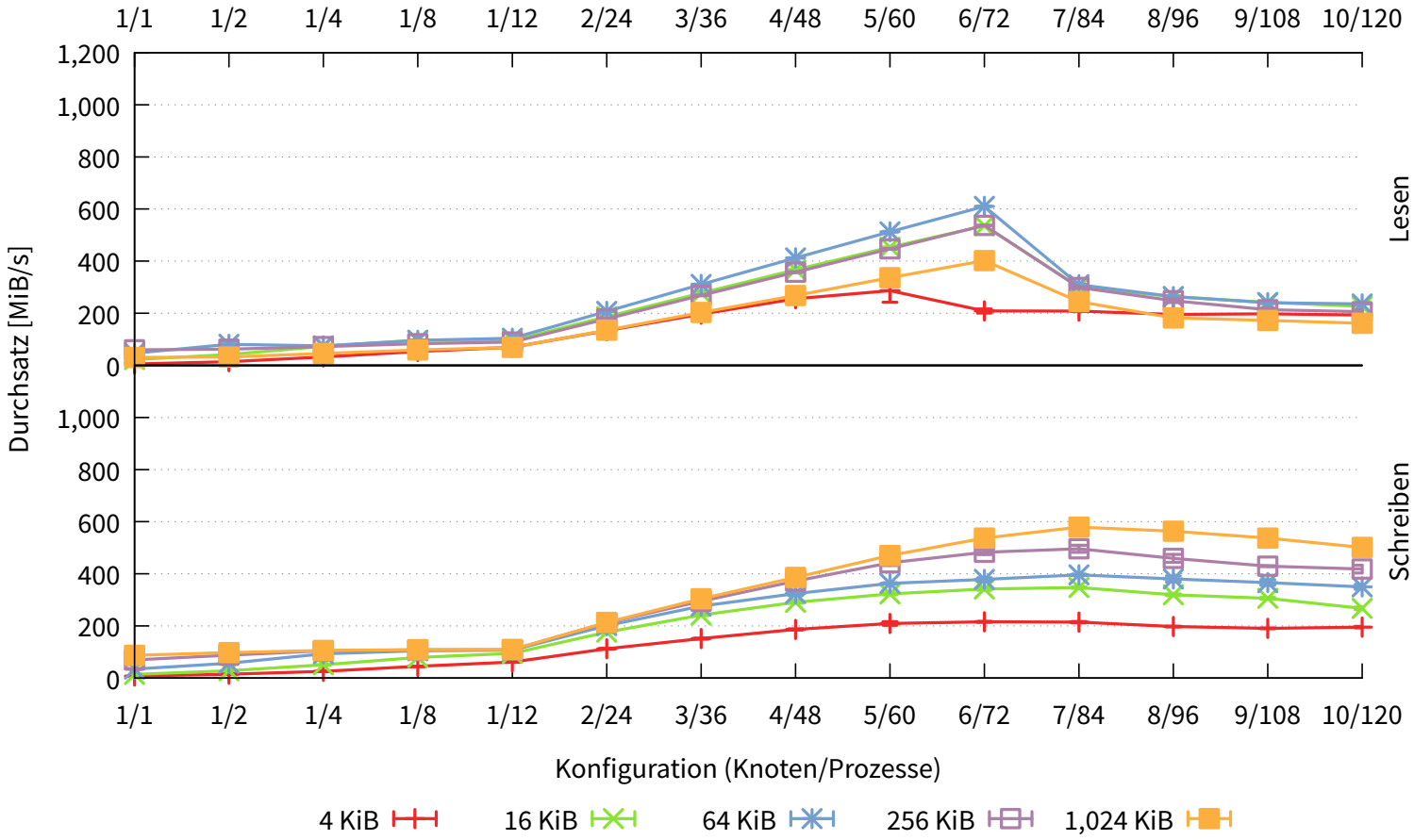


Abbildung: OrangeFS mit prozess-lokalen Dateien

OrangeFS...

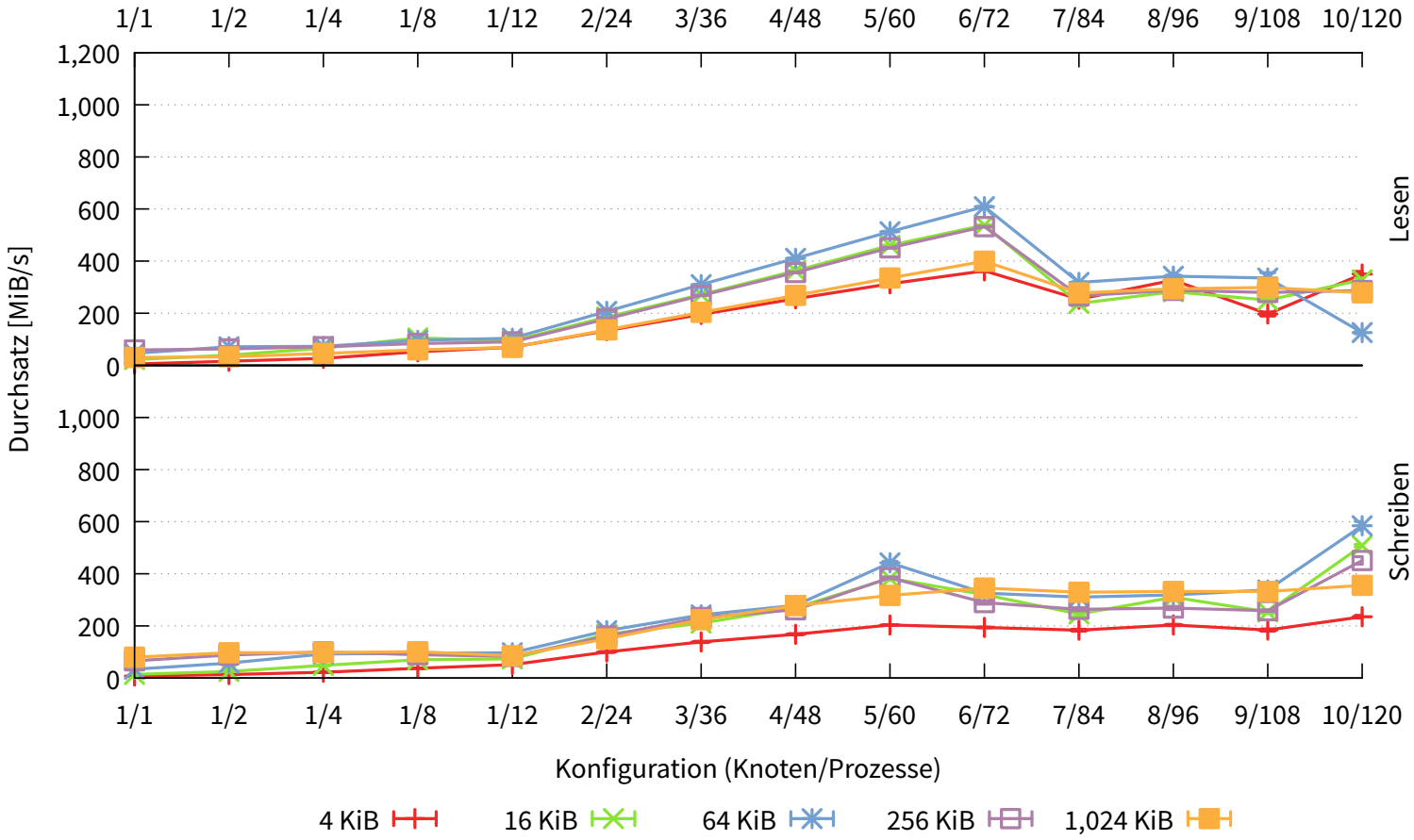


Abbildung: OrangeFS mit gemeinsamer Datei

OrangeFS…

- OrangeFS ist optimiert für nicht-konfligierende Zugriffe
  - Dadurch keine Sperren notwendig
  - Hohe Leistung auch bei gemeinsamen Dateien
- Leistungsprobleme durch darunter liegendes Dateisystem
- Außerdem geringe Metadatenleistung

Lustre

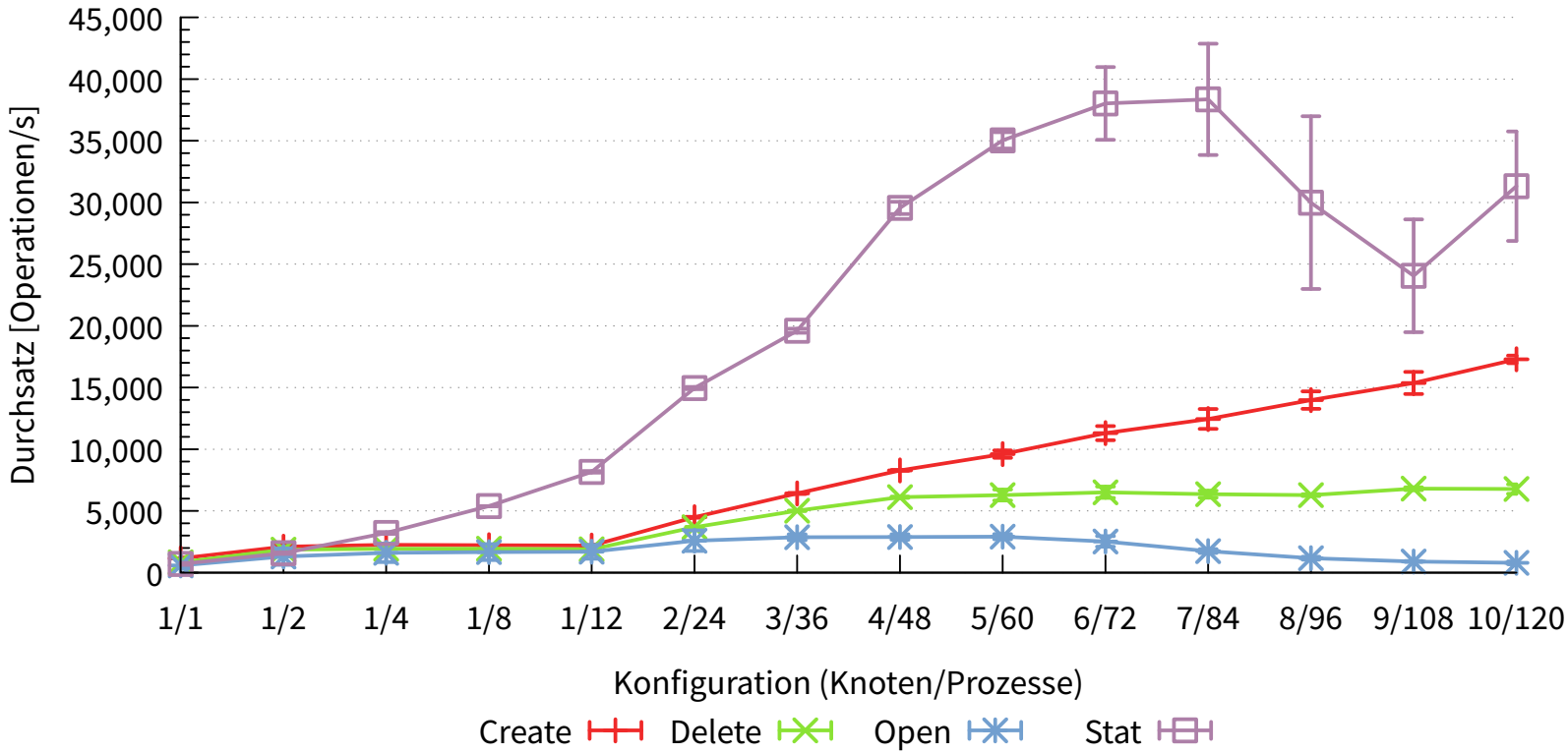


Abbildung: Lustre mit prozess-lokalen Verzeichnissen

Lustre…

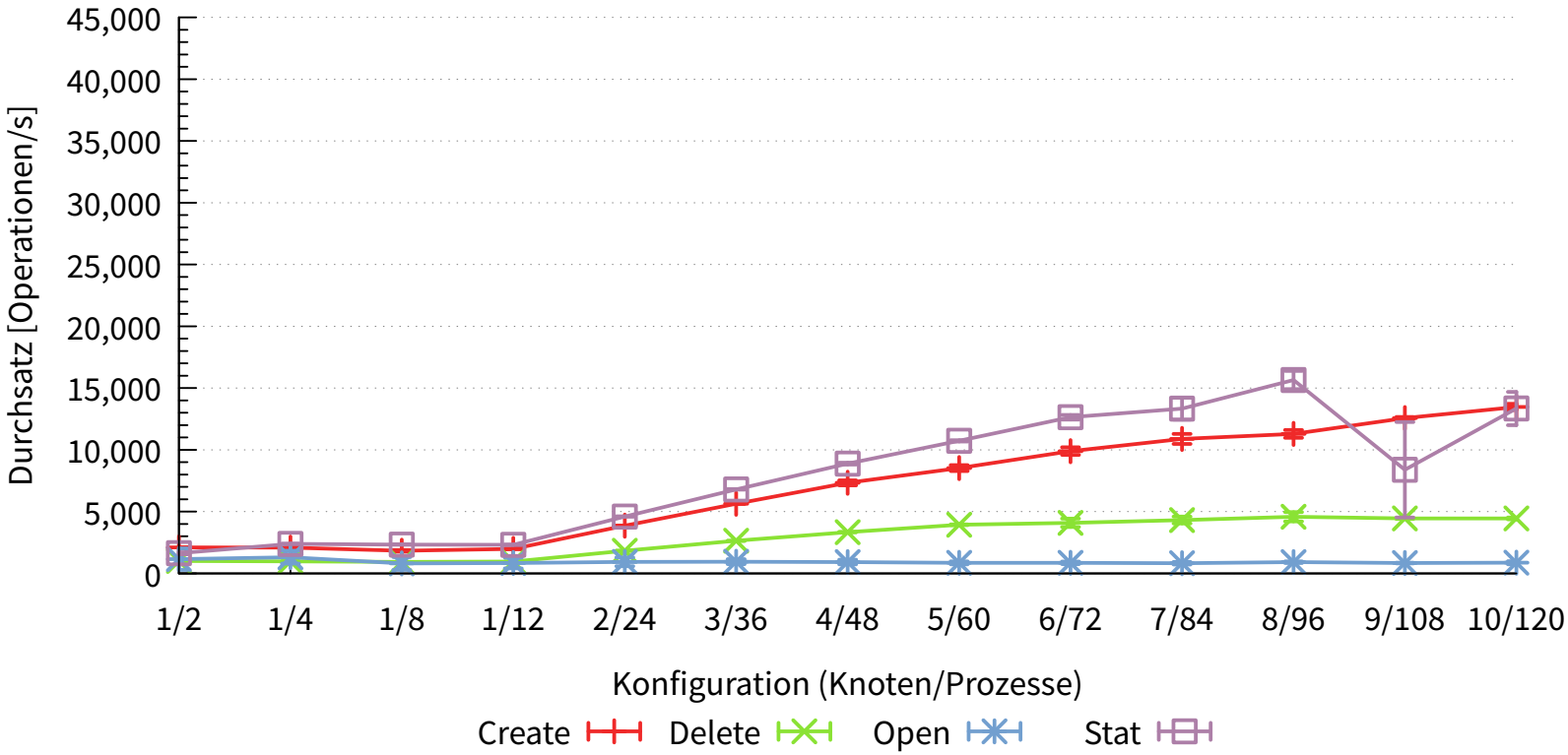


Abbildung: Lustre mit gemeinsamem Verzeichnis

Lustre…

- Deutliche Unterschiede zwischen prozess-lokalen und gemeinsamen Verzeichnissen
- Passable Leistung für Dateierzeugung
- Leistungsprobleme beim Öffnen und Löschen
  - Interessanterweise ist Öffnen langsamer als Erzeugen
- Leistung nur bedingt verbesserbar [1]
  - Schnellere Speicherhardware kann nicht voll ausgenutzt werden
  - Zusätzliche Sockel verringern manchmal sogar die Leistung

Ausblick

- E/A-Bibliotheken bieten Komfortfunktionen auf Basis von parallelen verteilen Dateisystemen
  - Z. B. selbst-beschreibende Daten mit NetCDF und HDF
- Semantiken haben großen Einfluss auf erreichbare Leistung
  - Forschungsthema: Dynamisch anpassbare Semantiken

Zusammenfassung

- Parallele verteilte Dateisysteme bieten gleichzeitigen Zugriff für viele Clients
  - Skalierbarer gleichzeitiger Zugriff schwierig
  - Verteilen außerdem Daten und Metadaten für erhöhten Durchsatz und Kapazität
- Üblicherweise Aufteilung in Daten- und Metadatenserver
- Zugriff über E/A-Schnittstelle
  - Häufig POSIX oder MPI-IO
- Wichtige Vertreter sind Lustre und GPFS
  - OrangeFS bietet alternativen Ansatz

1 Parallele verteilte Dateisysteme
- Orientierung
- Konzepte
- Leistungsüberlegungen
- Lustre
- OrangeFS
- Leistungsanalyse
- Ausblick und Zusammenfassung

2 Quellen

Quellen I

[1] Konstantinos Chasapis, Manuel Dolz, Michael Kuhn, and Thomas Ludwig. Evaluating Lustre's Metadata Server on a Multi-socket Platform. In *Proceedings of the 9th Parallel Data Storage Workshop*, number 2014 in PDSW, pages 13–18, Piscataway, NJ, USA, 2014. IEEE Press.

[2] OpenSFS and EOFS. Lustre. http://lustre.org/.

[3] OrangeFS Development Team. OrangeFS. http://www.orangefs.org/.

[4] Wikipedia. IOPS. http://en.wikipedia.org/wiki/IOPS.