Parallele verteilte Dateisysteme

Hochleistungs-Ein-/Ausgabe

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Michael Kuhn

2019-04-23

Wissenschaftliches Rechnen
Fachbereich Informatik
Universität Hamburg

Parallele verteilte Dateisysteme

E/A-Schichten

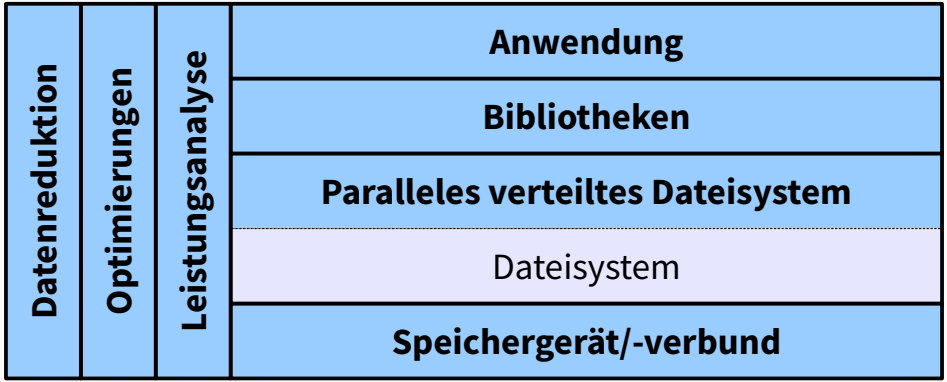


Abbildung 1: E/A-Schichten und orthogonale Themen

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 üblicherweise keine vollständige Sicht
- Benennung uneinheitlich
  - Oft nur „paralleles Dateisystem“ oder „Clusterdateisystem“

Definition...

- Im Hochleistungsrechnen üblicherweise parallele verteilte Dateisysteme
  - Insbesondere für die Ein-/Ausgabe von Anwendungsdaten
  - NFS nur für nicht leistungskritische Anwendungen (z. B. Home-Verzeichnisse)
- Lokale Dateisysteme erlauben auch parallelen Zugriff
  - Sperren z. B. via `flock` oder `lockf`
- Verteilung erfordert entsprechende Architektur und weiteren Zusatzaufwand
  - Je nach E/A-Schnittstelle 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 für komfortableren Zugriff
  - Stellt direkt ein Dateisystem bereit
    - Üblicherweise NFS oder SMB

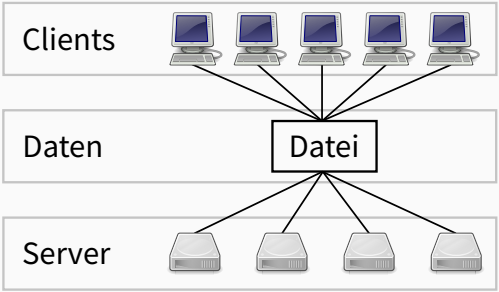


Abbildung 2: Paralleler Zugriff und Datenverteilung

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

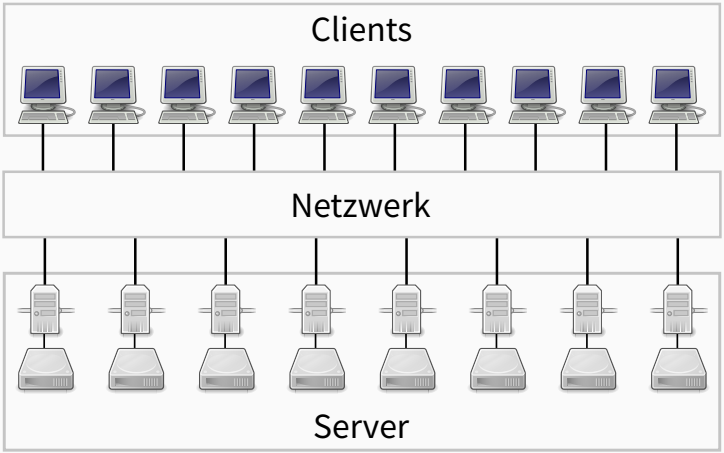


Abbildung 3: Paralleles verteiltes Dateisystem

Architektur...

- Trennung in Clients und Server
  - Spezialisierung auf jeweilige Funktionalität
  - Keine gegenseitige Beeinflussung
- Clients führen parallele Anwendungen aus
  - Häufig nur lokaler Speicher für Betriebssystem und/oder Caching
  - Haben über das Netzwerk Zugriff auf Dateisystem
    - Üblicherweise kein direkter Zugriff auf Speichergeräte der Server
- Trennung in Daten- und Metadatenserver
  - Sinnvoll aufgrund unterschiedlicher Zugriffsmuster
    - Datendurchsatz vs. Anfragendurchsatz
  - Bereitstellung durch „einfache“ Controller oder vollwertige Server

Architektur...

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

Architektur...

1. Clients kennen zuständigen Server
   - Vorteile: Einfacheres Kommunikationsprotokoll durch direkte Client-Server-Kommunikation, keine Kommunikation zwischen Servern
   - Nachteile: Verteilungslogik muss von Clients implementiert werden, zusätzliche clientseitige Informationen notwendig
2. Clients kontaktieren einen zufälligen Server
   - Vorteile: Clients müssen Daten-/Metadatenverteilung nicht kennen, Last-Balancierung und Replikation einfacher umzusetzen
   - Nachteile: Höhere Latenz durch zusätzliche Nachrichten, komplexeres und fehleranfälligeres 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...

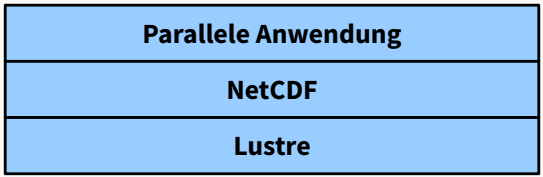


Abbildung 4: E/A-Stack aus Anwendungssicht

- Anwendungen nutzen höher abstrahierende Schnittstellen
  - NetCDF bietet selbst-beschreibendes Datenformat
- Paralleles verteiltes Dateisystem kümmert sich um effizienten Zugriff und Verteiltung
  - Optimalerweise kein Wissen über konkrete Implementierung notwendig

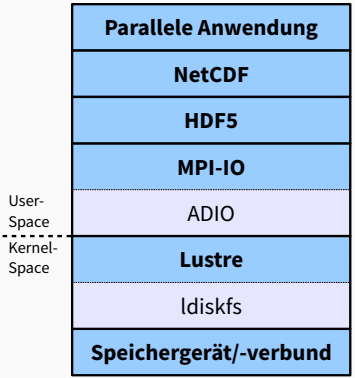
Semantik

- 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 direkt sichtbar
    - Zuerst nur innerhalb der eigenen Sitzung
  - Für andere Clients erst nach Sitzungsende
    - `close` schreibt Änderungen und liefert eventuelle Fehler
- Später: MPI-IO
  - Weniger strikt für höhere Skalierbarkeit

Realität



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

Abbildung 5: E/A-Stack im HPC

Realität...

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

Abbildung 6: Abstraktionsstufen

- Hohe Abstraktion bietet hohen Komfort aber wenig Kontrolle
  - „Schreibe Matrix `m`“
- Niedrige Abstraktion erlaubt genaues Tuning
  - „Schreibe `n` Bytes an Offset `m` mit synchroner E/A“

Beispiele

- Spectrum Scale (IBM, früher GPFS)
- 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.
  - Möglichst asynchron, damit der Prozessor nicht auf Daten warten muss
- Parallele verteilte E/A muss über Netzwerk abgewickelt werden
  - Einschränkungen bezüglich Durchsatz und Latenz
- Ausblick: Neuartige Konzepte wie Burst-Buffer
  - Nehmen Daten temporär auf und reichen an das Dateisystem weiter
  - Z. B. SSD-Knoten oder knoten-lokale nichtflüchtige Speicher (NVRAM/SSD)
- Sowohl Daten- als auch Metadatenleistung kann problematisch sein

- Daten werden potentiell von mehreren Clients zugegriffen
  - Lesezugriffe meist unproblematisch, da keine Konflikte auftreten
  - Bei Schreibzugriffen Unterscheidung in überlappend und nicht überlappend
- Überlappende Schreibzugriffe
  - Benötigt üblicherweise Sperren, daher häufig verteilte Sperrenverwaltung
  - Leistung stark abhängig von der E/A-Semantik
    - Korrekte Abarbeitung in POSIX, undefiniertes Verhalten in MPI-IO
- Nicht überlappende Schreibzugriffe
  - Potentielle Leistungsprobleme durch grobgranulare Sperren

- Datenverteilung auch relevant für Leistung
  - Insbesondere bei nicht überlappenden Zugriffen
    - Unterschiedliche Clients sollten auf unterschiedliche Server zugreifen
- Realisiert über Verteilungsfunktionen
  - Üblicherweise simples Round-Robin aber durch Benutzer anpassbar
  - 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

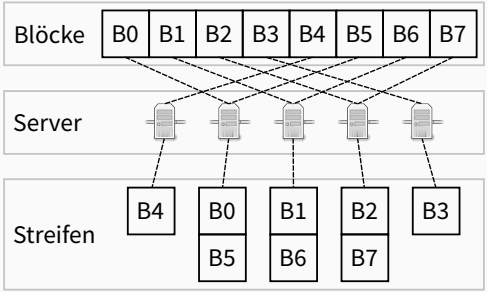


Abbildung 7: Round-Robin-Datenverteilung

- Datei wird in Blöcke unterteilt und auf Server verteilt
  - In diesem Fall acht Blöcke verteilt auf fünf Streifen
- Verteiltung muss nicht auf dem ersten Server starten

- Metadaten werden von mehreren Clients zugegriffen
  - Lesezugriffe üblicherweise wieder unproblematisch
  - Parallele Änderungen an Größe, Zeitstempel etc. führen zu Konflikten
- Metadaten für eine Datei von einem Server verwaltet
  - Aktualisierungen daher meist auch seriell
  - Metadaten einer Datei können nicht sinnvoll auf mehrere Server verteilt werden
- Potentiell mehrere Millionen Clients
  - Distributed Denial of Service durch Clients auf Metadatenserver

Metadaten...

- Ansatz zur effizienten verteilten Metadatenverwaltung
  - Einige Metadaten werden häufig aktualisiert
    - Z. B. Größe und Zeitstempel
  - Nicht zentral speichern, sondern zur Laufzeit berechnen
    - 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 wie die Pfadauflösung sind inhärent seriell
  - Laut POSIX muss jede Pfadkomponente geprüft werden

1. **/**foo/bar
   1.1 Inode des Wurzelverzeichnisses lesen
   1.2 Zugriffsrechte überprüfen und bei Bedarf Fehler zurückgeben
   1.3 Wurzelverzeichnis lesen und nach foo durchsuchen
2. /**foo/**bar
   2.1 Inode des Verzeichnisses lesen
   2.2 Zugriffsrechte überprüfen und bei Bedarf Fehler zurückgeben
   2.3 Verzeichnis lesen und nach bar durchsuchen
3. /foo/**bar**
   3.1 Inode der Datei lesen
   3.2 Zugriffsrechte überprüfen und bei Bedarf Fehler zurückgeben
   3.3 Auf Datei zugreifen

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

Tabelle 1: 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
- Mögliches Problem: Software muss Leistung auch ausnutzen können

Leistungsgrößen

- Parallele verteilte Dateisysteme große Speichersysteme möglich
- 2009: Blizzard am DKRZ (GPFS)
  - Größe: 7 PB
  - Durchsatz: 30 GB/s
- 2012: Titan am ORNL (Lustre)
  - Größe: 40 PB
  - Durchsatz: 1,4 TB/s
- 2015/2016: Mistral am DKRZ (Lustre)
  - Größe: 54 PiB
  - Durchsatz: 450 GB/s (5,9 GB/s pro Knoten)
  - IOPS: 80.000 Operationen/s (nur Phase 1)

- Eines der bekanntesten parallelen verteilten Dateisysteme
- Open Source (GPLv2)
  - $>$ 550.000 Zeilen Code
- Unterstützung für Linux
  - Name abgeleitet von Linux und Cluster
  - Kernfunktionalität vollständig im Kernel implementiert
- 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
  - DDN/Whamcloud, Intel, Seagate, 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)
  - Unterstützung für 16 MiB große RPCs
  - Unterstützung für `ladvise`
- Version 2.10 (Juni 2017)
  - Dateisystemschnappschüsse
  - Projekt-Quotas
  - Progressive Datei-Layouts
- Version 2.11 (April 2018)
  - Daten auf Metadatenservern
  - Dateibasierte Redundanz
  - Lock Ahead

```
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 Verzeichnisse auf unterschiedliche Metadatenserver zu verteilen
  - `/scratch` für große, `/home` für viele kleinere Dateien
  - Statischer Ansatz, Verteilung muss manuell festgelegt werden
- Unterstützung für verteilte Verzeichnisse
  - `/striped` über drei Metadatenserver verteilt

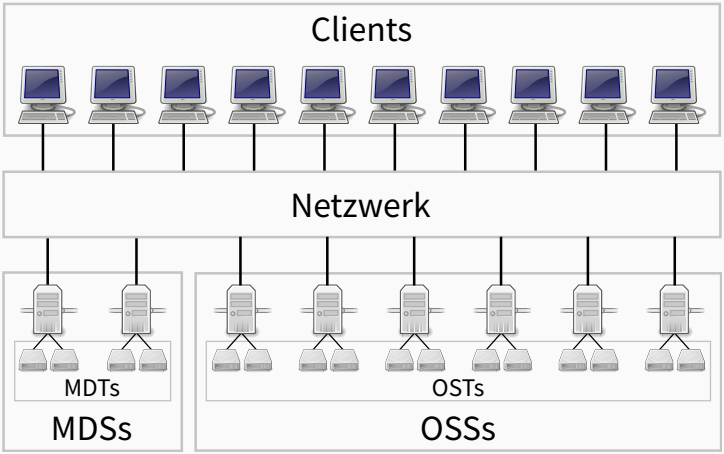


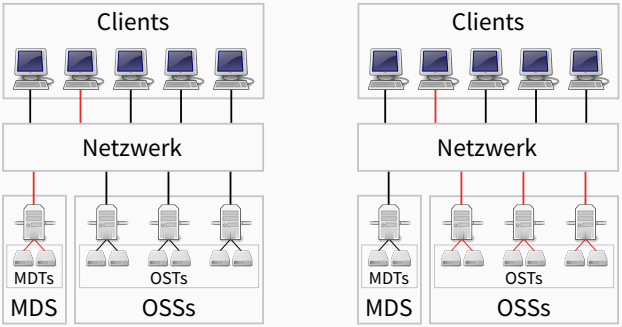
Abbildung 8: Lustre-Architektur

- Object Storage Servers (OSSs)
  - Verwalten Daten in Form von Objekten mit Zugriff auf Byte-Ebene
  - Speicherung auf Object Storage Targets (OSTs)
    - Ein Target kann beispielsweise ein RAID-Array sein
- Metadata Servers (MDSs)
  - Verwalten Metadaten und sind nicht in die eigentliche E/A involviert
  - Speicherung auf Metadata Targets (MDTs)
    - Kann entsprechend dem Zugriffsmuster z. B. als SSD-RAID ausgelegt werden
- Verteilung über Targets, nicht über Server
  - Ein Server kann mehrere Targets verwalten
  - Eventuell keine Leistungsvorteile, wenn genutzte Targets auf gleichem Server liegen

- Daten- und Metadatenserver nutzen lokales Dateisystem
  - Üblicherweise ldiskfs (ext4-Fork)
    - Führt unter anderem zu doppeltem POSIX-Overhead
    - Außerdem keine Unterstützung für Prüfsummen, Kompression, Schnappschüsse etc.
  - Alternativ ZFS
    - Direkter Zugriff auf Data Management Unit und Adaptive Replacement Cache
- Kein direkter Zugriff auf Speichergeräte durch Clients
  1. Clients senden Anfragen an Server
  2. Server führen Operationen durch
  3. Server schicken Ergebnisse an Clients

- Lustre nutzt ein direktes Kommunikationsprotokoll
  - Clients kontaktieren Managementserver und kennen danach zuständige Server
- Server müssen sich nicht gegenseitig kennen
  - Server müssen Managementserver kennen und sich dort an-/abmelden
  - 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 Verteilungsfunktion und zuständige Datenserver
3. Clients kontaktieren zuständige Datenserver



(a) Metadatenzugriff **(b)** Datenzugriff

(a) Metadatenserverzugriff nur für initiales Öffnen
(b) Danach direkter paralleler Zugriff auf Datenserver
- Vorausgesetzt es werden keine Metadaten geändert

Unterstützung

- Lustre ist ein Kernel-Dateisystem
  - Sowohl Client als auch Server
- Client unterstützt (relativ) aktuelle Kernel
  - Unterstützung neuer Kernel dauert manchmal eine Weile
  - Erfordert Nutzung neuer Lustre-Versionen, die evtl. keine Langzeitunterstützung haben
- Server unterstützt nur ausgewählte Enterprise-Kernel
  - Red Hat Enterprise Linux (oder CentOS) und seit Neuestem Ubuntu
  - Hauptsächlich verursacht durch ldiskfs
    - Inzwischen auch als “patchless”-Variante
    - Mit ZFS auch Unterstützung des Standard-Kernels

- 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
    - Allerdings eigenes Modul in ROMIOs ADIO-Schicht

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

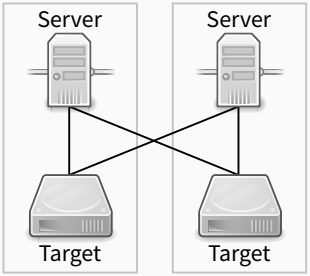


Abbildung 9: Aktiv/Aktiv-Failover-Konfiguration

- Server können jeweils Rolle des anderen übernehmen
  - Im Fehlerfall verwaltet dann ein Server temporär zwei Targets
- Kann auch für unterbrechungsfreie Upgrades genutzt werden

- 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
  - POSIX-Bibliotheken oder FUSE-Dateisystem
  - Alternativ optionales Kernelmodul

Funktionalität...

- OrangeFS ist nicht POSIX-konform
  - 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

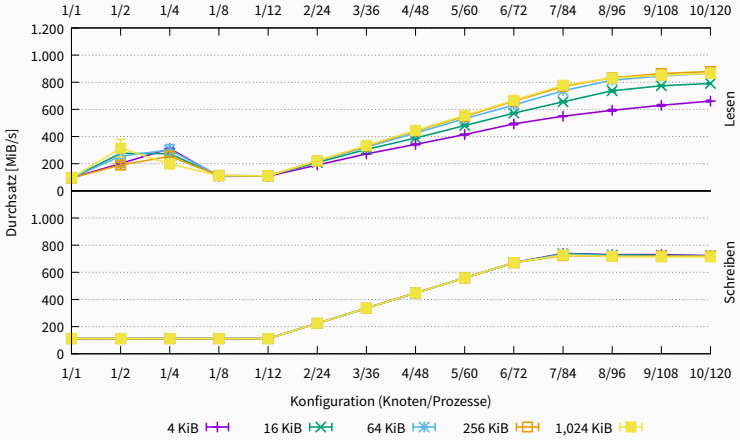


Abbildung 10: Lustre 2.5 mit prozess-lokalen Dateien

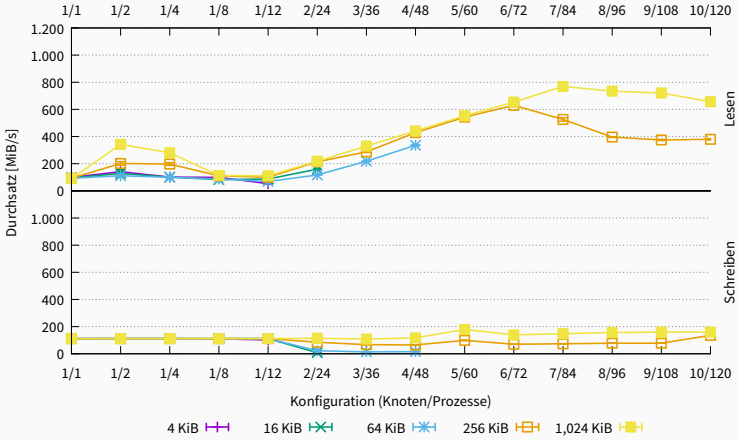


Abbildung 11: Lustre 2.5 mit gemeinsamer Datei

- 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 Read Ahead und Lock Ahead 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

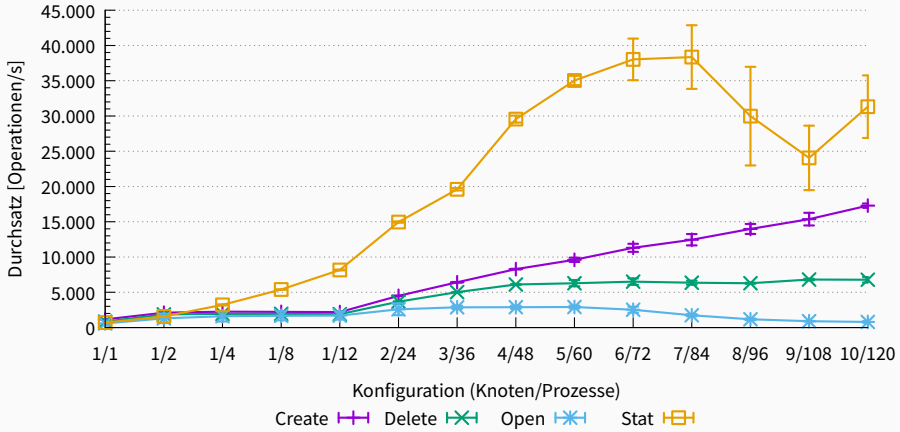


Abbildung 12: Lustre 2.5 mit prozess-lokalen Verzeichnissen

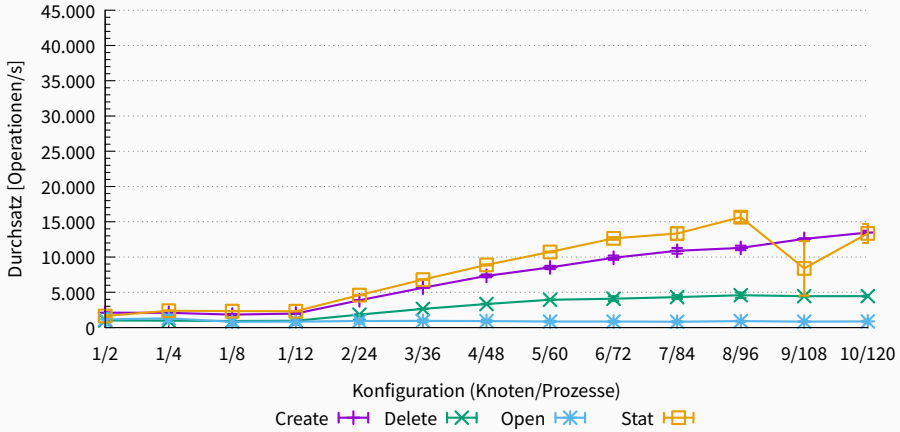


Abbildung 13: Lustre 2.5 mit gemeinsamem Verzeichnis

- 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

- Effiziente Nutzung paralleler verteilter Dateisystem kann schwierig sein
  - Genaue Kenntnis über Leistungsverhalten notwendig
    - Insbesondere bei POSIX-konformen Dateisystemen
  - E/A-Bibliotheken bieten Optimierungen und 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
  - Verteilen außerdem Daten und Metadaten für erhöhten Durchsatz und Kapazität
- Üblicherweise Aufteilung in Daten- und Metadatenserver
  - Unterschiedliche Zugriffsmuster erfordern unterschiedliche Optimierungen
- Zugriff über E/A-Schnittstelle
  - Häufig POSIX oder MPI-IO
- Wichtige Vertreter sind Lustre und Spectrum Scale
  - OrangeFS bietet alternativen Ansatz

Quellen

[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`.