HAW HAMBURG

Bachelorarbeit

Alexander Mendel

# Aufbau eines verteilten Dateisystems zur Beschleunigung von Deep Learning Anwendungen

*Fakultät Technik und Informatik*
*Department Informatik*

*Faculty of Engineering and Computer Science*
*Department Computer Science*

Alexander Mendel

# Aufbau eines verteilten Dateisystems zur Beschleunigung von Deep Learning Anwendungen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Informatik Technischer Systeme*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Dr. Tobias Eichler

Eingereicht am: 11. Januar 2024

**Alexander Mendel**

**Thema der Arbeit**

Aufbau eines verteilten Dateisystems zur Beschleunigung von Deep Learning Anwendungen

**Stichworte**

Verteiltes Dateisystem, Rechenzentrum, Deep Learning, Machine Learning, High Performance Computing, parallel Computing, Cluster, Administration

**Kurzzusammenfassung**

In modernen Rechenzentren werden mit zunehmender Rechenleistung schnelle Dateisysteme benötigt. Zur Beschleunigung von Deep Learning Anwendungen soll in dieser Arbeit eine geeignete Konfiguration für ein verteiltes Dateisystem gefunden werden. Dabei werden verschiedene Konfigurationen evaluiert. ...

**Alexander Mendel**

**Title of Thesis**

Distributed file system setup to accelerate deep learning applications

**Keywords**

distributed file system, data center, deep learning, machine learning, high performance Computing, parallel computing, cluster, administration

**Abstract**

Modern data centers with high processing capabilities require fast storage systems. This thesis examines distributed filesystems in search for a suitable configuration to accelerate deep learning tasks. ...

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# 1 Einleitung

Heutzutage sind Anwendungen, bei denen Machine Learning zum Einsatz kommt, immer mehr verbreitet. Machine Learning ist ein Teilgebiet der künstlichen Intelligenz, bei dem Computer Muster aus Daten verarbeiten und daraus lernen, ohne explizite Programmierung dafür zu erfordern.

In den letzten Jahren haben Machine Learning Technologien begonnen, einen großen Einfluss auf viele Branchen in verschiedenen Bereichen zu nehmen. Dazu gehören zum Beispiel Themenfelder aus der Wirtschaft und dem Gesundheitswesen oder Anwendungsgebiete in der Analyse von Daten in verschiedenen Formen, wie Sprache, Bild oder Finanzen [23]. Auch im Bereich der Forschung oder Kunst gibt es viele neue Ansätze, bei denen Machine Learning eingesetzt wird und für neue Erkenntnisse sorgt. Unternehmen können damit Vorteile gegenüber Konkurrenten ohne solche Prozesse erzielen. Viele der Aufgaben können mit anderen Prozessen nur schwer oder gar nicht erzielt werden [25].

Anwendungen aus dem Bereich Machine Learning prägen auch zunehmend die Entwicklung und den Aufbau von Rechenzentren. Die Anbieter von großen Cloud-Computing Rechenzentren bieten spezielle Systeme für Anwendungsfälle aus dem Bereich Machine Learning an [24]. Die Konstruktion eines Rechenzentrums bringt mehrere Herausforderungen mit sich. Es müssen beim Aufbau spezielle Hardware und Software ausgewählt und konfiguriert werden, um den Anforderungen von großen Datenbeständen und langlaufenden Prozessen gerecht zu werden.

Im Folgenden wird die Problematik aus einem konkreten Einsatzszenario dieser Art dargestellt und das daraus resultierende Thema dieser Arbeit einleitend erklärt.

## 1.1 Problemanalyse

Viele Machine Learning Anwendungen haben gemeinsam, dass sehr viel parallele Rechenleistung benötigt wird. Ein Grafikprozessor (GPU) bietet, im Gegensatz zum Haupt-

prozessor (CPU), eine hohe Anzahl an Rechenkernen, welche Aufgaben durch parallele Ausführung stark beschleunigen können [6]. Aktuelle GPUs umfassen mehrere tausende Rechenkerne pro Erweiterungskarte [28], wohingegen Prozessoren mit x86-64 Architektur (RISC-Befehlssatz) aktuell maximal hunderte Rechenkerne pro CPU unterstützen [2]. Machine Learning Anwendungen, bei denen große Datenmengen verarbeitet werden, können für die parallele Ausführung auf GPUs mit einer proprietären Befehlsarchitektur optimiert werden und Berechnungen effizienter und schneller als CPUs ausführen [36]. GPUs kommen daher vermehrt als Coprozessoren in Servern großer Rechenzentren zur Beschleunigung von Anwendungen und Berechnungen zum Einsatz.

Besonders die Themenfelder von Deep Learning, einem Teilfeld von Machine Learning, benötigen viel Rechenkapazität. Die zugehörigen Anwendungsfälle wie Spracherkennung und Bilderkennung bringen mit sich, dass große Datensätze (Datasets) mit Bild[1]- oder Audiodateien[2] verwendet werden, die viel Speicherplatz mit schneller Anbindung benötigen, auf die ein GPU-Rechenserver zugreifen muss. Mit steigender Rechenleistung durch mehrere parallel rechnende GPUs steigt auch der Leistungsbedarf an ein schnelles Dateisystem, auf dem Daten gespeichert werden.

Neben der Optimierung der Rechenleistung durch GPUs bringt die Optimierung der Leistung des Dateisystems potenziell weitere Optimierung der Gesamtrechenleistung mit sich.

Damit das System effizient ist, möchte man vermeiden, dass durch ein zu langsames Dateisystem die GPUs nicht schnell genug Daten lesen können und so nicht die volle Rechenleistung genutzt werden kann.

Vergleichbar mit der typischen Speicherhierarchie (vgl. Tanenbaum Kapitel 1.3.2 - [41]) ist Speicher, der möglichst schnell, groß und günstig ist, wünschenswert, in der realen Welt aber nicht existent.

Nonvolatile Memory Express-Laufwerke (NVMe) sind für diese Anforderungen ein guter Kompromiss und bieten eine geeignete Grundlage für ein schnelles Dateisystem (vgl. [38]). Die Anbindung über PCIe ermöglicht die direkte Kommunikation zwischen Laufwerk und CPU, gegenüber der SATA- oder SAS-Schnittstelle (vgl. [5]). Die Latenz ist dadurch am geringsten und der Durchsatz kann durch die Bandbreite der Anbindung höher sein.

[1] vgl. Bild - Dataset “MS COCO” [12, 22] und bewegtes Bild - Dataset “UCF101” [39, 31]
[2] vgl. Paper [19, 34] mit Dataset “Librispeech” [9, 40]

Lokale NVMe-Laufwerke als Dateisystem bzw. Cache in GPU-Servern sind ein gängiger Ansatz, um Prozesse zu beschleunigen und Verzögerung zu reduzieren (vgl. [3] und [37]). Dieser Ansatz ist bei Verwendung eines GPU-Servers eine praktikable Lösung. In Rechenzentren gibt es Anwendungsszenarien, bei denen die Berechnung parallel über mehrere GPU-Server verteilt ausgeführt werden. In dem Fall sind die lokalen Laufwerke nicht synchronisiert.

Ein **verteiltes Dateisystem**, welches horizontal skaliert (vgl. Tanenbaum Kapitel 2.2.2 - [42]), bietet die Möglichkeit den Speicher auf mehrere Server zu verteilen und sorgt für parallelen und jederzeit synchronen Zugriff auf das Dateisystem.

So kann in einem Rechenzentrum ein Verbund an GPU-Servern wachsen und ebenso das Dateisystem für die Anwendungsdaten mehr Speicherkapazität und Parallelität bieten. Dieser Lösungsansatz bietet einen hohen Grad an Flexibilität für die Architektur eines wachsenden Rechenzentrums.

## 1.2 Vorhandenes System

An der Hochschule für Angewandte Wissenschaften Hamburg (HAW Hamburg) betreibt seit 2016 das Forschungs- und Transferzentrum Smart Systems (FTZ SMSY) eine Forschungsinfrastruktur. Diese biete eine Computing-Cloud-Lösung für Forschungs- und Projektarbeiten von Studierenden und Externen. Im Rechenzentrum werden mehrere Compute-Server unter anderen einige mit Grafikkarten (GPU-Server) für Machine-Learning-Anwendungen betrieben. Die Anwender des Systems, bspw. aus der Machine Learning-AG, können über ein Buchungssystem [18] einen der GPU-Server für ein Projekt reservieren.

In Abbildung 1.1 ist der aktuelle logische Aufbau des Rechenzentrums dargestellt. Nachdem die Projektgruppe den Anwender als *Benutzer* hinzugefügt hat, können sich diese über ein VPN mit einer virtuellen Maschine (VM-Client) auf dem gebuchten Compute-Server bzw. GPU-Server verbinden. Ein dedizierter Dateiserver (NFS-Server) fungiert als Massenspeicher für die virtuellen Maschinen. Eine Freigabe des NFS-Servers ist auf dem VM-Client eingehängt und dient als Zwischenspeicher für Datasets, die für Berechnungen der GPU-Server benötigt werden.

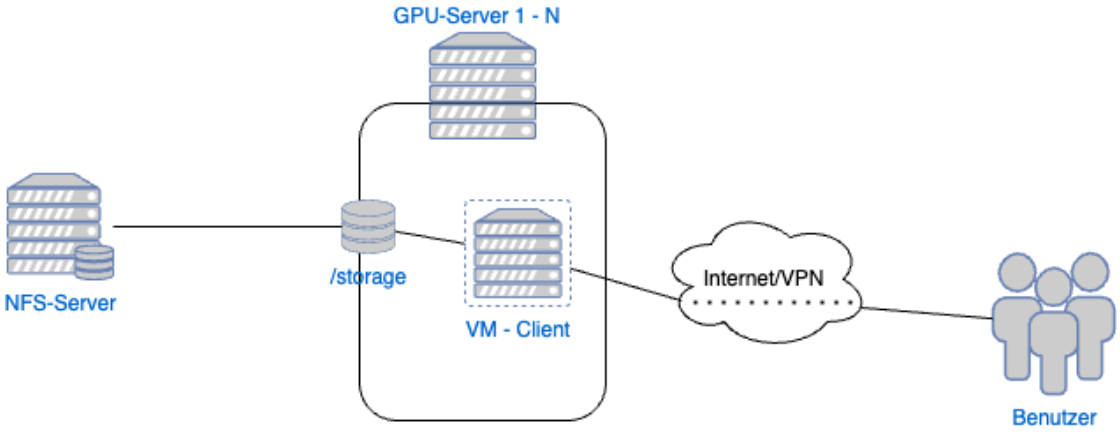


Abbildung 1.1: Aktuelle Aufbaustruktur des Rechenzentrums
Quelle: Eigene Darstellung

Der Anwender kann die Daten für ein Projekt auf dem eingehängten NFS-Dateisystem speichern. Während die Berechnung läuft, laden die GPUs die Daten über die eingehängte Freigabe vom NFS-Server.

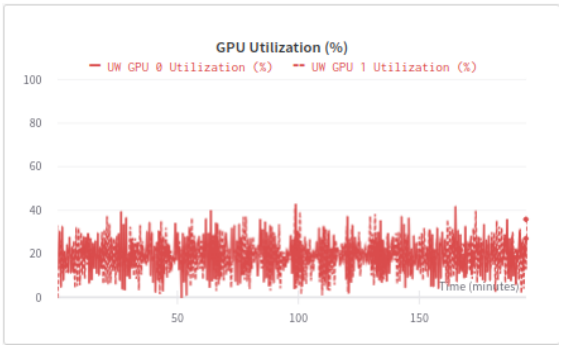


Abbildung 1.2: GPU-Auslastung während datenintensivem Training
Quelle: J. Zach

Wie Abbildung 1.2 darstellt, können die Nutzer der GPU-Server derzeit besonders bei datenintensiven Anwendungen beobachten, dass während Berechnungen auch über längere Zeit die GPU-Auslastung signifikant abfällt (J. Zach und F. Voigt, persönliche Kommunikation, 14. Dezember 2023). Die Übertragungsgeschwindigkeit und Leistung des NFS-Servers kann einer der maßgebenden Faktoren für diese Beobachtungen sein.

Dem können Anwender teilweise entgegenwirken, indem die Batch-Größe in den Anwendungsfällen mit Deep Learning angepasst wird. Eine höhere Batch-Größe kann allerdings zu ungenaueren Ergebnissen führen [27].

Zur Optimierung dieser beobachteten Einschränkungen im Betrieb hat das FTZ SMSY Hardware bestehend aus vier Servern mit mehreren NVMe-Laufwerken eingekauft, die als Dateisystem konfiguriert werden sollen.

## 1.3 Zielsetzung

Ziel ist, ein verteiltes Dateisystem auf Servern mit NVMe-Laufwerken zu konfigurieren, um dieses in die vorhandene Infrastruktur einzubinden und als Dateisystem mit der Funktion als schneller Zwischenspeicher für die GPU-Server bereitzustellen (vgl. Abbildung 1.3). Das verteilte Dateisystem soll auf den Compute- bzw. GPU-Servern eingebunden werden und exklusiv als schneller Zwischenspeicher für Deep Learning Anwendugnen verwendet werden. Die GPU-Server sollen mit möglichst hoher Geschwindigkeit Daten verarbeiten können, so dass die Rechenkapazität bei datenintensiven Anwendungen verbessert wird.

Für die Installation des Systems soll durch Testen verschiedener Ansätze die optimale Konfiguration für die Anwendung gefunden werden. Die Installation und Konfiguration des Systems soll größtenteils mit Skripten automatisiert werden, um das gesamte System für das Team des Rechenzentrums (Projektgruppe) wartbar zu machen.

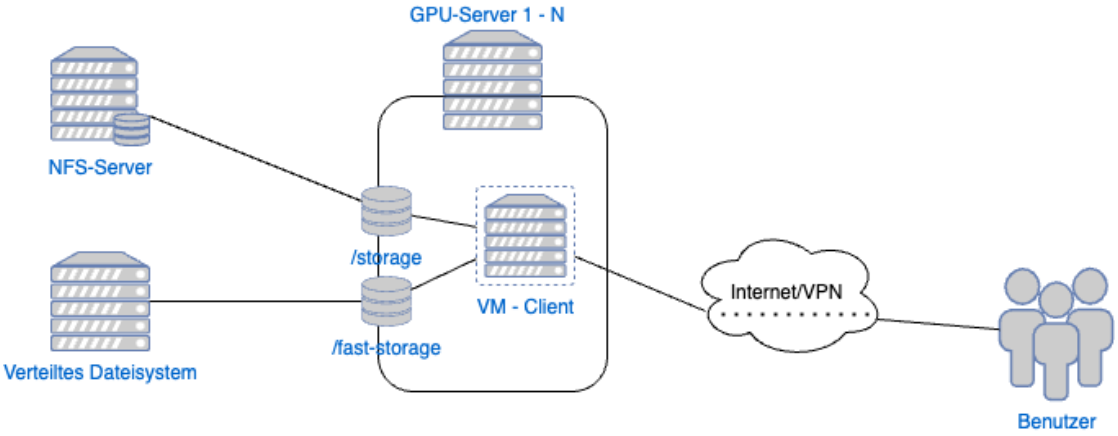


Abbildung 1.3: Logischer Aufbau des Rechenzentrums mit verteiltem Dateisystem
Quelle: Eigene Darstellung

## 1.4 Gliederung der Arbeit

Die Arbeit ist in fünf Kapitel aufgeteilt. Im 1. Kapitel, der Einleitung, wurde das Problem geschildert und der Lösungsansatz beschrieben. Es wurden das bestehende System des FTZ und die Ergänzung des Systems durch ein verteiltes Dateisystem dargestellt.

In Kapitel 2 werden die Anforderungen an das System analysiert, das ausgewählte System beschrieben und der Ansatz für das Testen des Systems erläutert, um verschiedene Konfigurationen zu vergleichen.

Kapitel 3 beschreibt den Ansatz bei der Installation, die Umsetzung der automatisierten Installation und das Vorgehen bei den Benchmarks.

Anschließend folgt in Kapitel 4 die Evaluation der Ergebnisse aus den Benchmarks.

Die Schlussfolgerung des Gesamtergebnisses und das Fazit folgen in Kapitel 5.

# 2 Analyse

Wie in der Zielsetzung (s. Abschnitt 1.3) beschrieben, soll für die Umsetzung ein verteiltes Dateisystem im Rechenzentrum installiert werden.

In der Analyse wird im ersten Abschnitt der genauere technische Aufbau des bestehenden Rechenzentrums erklärt und beschrieben, wie die neuen Server integriert werden sollen. Anschließend werden in Abschnitt 2.2 verteilte Dateisysteme vorgestellt und die Anforderungen anhand von geeigneten Systemen erklärt und analysiert. Nach der Anforderungsanalyse und Wahl des verteilten Dateisystems (s. Abschnitt 2.3) folgt in Abschnitt 2.4 die Vorstellung der Architektur des für die Umsetzung gewählten Systems. Es werden die möglichen Konfigurationsansätze beschrieben und spezifische Features erklärt. In Abschnitt 2.5 wird der Ansatz für das Ausführen von Benchmarks auf dem System beschrieben, die für die Evaluation der Konfigurationen durchgeführt werden sollen.

## 2.1 Technischer Aufbau

Wie im Abschnitt 1.2 einführend beschrieben wurde, betreibt eine Projektgruppe des FTZ SMSY an der HAW ein Rechenzentrum mit u. a. GPU-Servern. Die Machine Learning AG ist eine aktive Nutzergruppe dieses Rechenzentrums. Für Projekte und Forschungsarbeiten reservieren sich die Anwender die Systeme, um auf den Servern Berechnungen durchzuführen.

Für das Vorhaben der Optimierung des Dateisystems, entsprechend Abschnitt 1.3, wurde Hardware bestehend aus vier Server-Nodes eingekauft, welche in die Infrastruktur eingebunden werden soll, um den GPU-Servern als schnelles Dateisystem zur Verfügung zu stehen.

In diesem Abschnitt wird der technische Aufbau der betroffenen Komponenten im Rechenzentrum erläutert und detaillierter beschrieben, mit welchen Komponenten die NVMe-Server konfiguriert sind. Dabei wird auch beschrieben, wie die Netzwerkverbindung zwischen NVMe-Servern und GPU-Servern aufgebaut ist.

### Server des Rechenzentrums

Die GPU-Server umfassen derzeit sechs Server mit Grafikkarten, die für Machine Learning Projekte verwendet werden können. Insgesamt stehen 20 NVIDIA Quadro P6000, 6 Tesla V100, 6 Tesla V100S und 4 Tesla A100 zur Verfügung.

Der dedizierte NFS-Server hat eine Gesamtkapazität von 600 TB und ist über eine Ethernet-Schnittstelle mit 10 Gb/s per iSCSI und NFS an die Compute- und GPU-Server bzw. die virtuellen Maschinen der Benutzer der Server angebunden. Der NFS-Server wird für mehr als 150 virtuelle Server als Laufwerk verwendet und als Datenablage zum Zwischenspeicher für Datasets für die GPU-Server. Der Speicher ist für verschiedene Anwendungsbereiche reserviert und wird auf den VM-Clients per NFS eingehängt.

### Neue NVMe-Server

Die Hardware für die NVMe-Server besteht aus einem 2U Supermicro SuperServer 2029BT-HNR. In dem Chassis sind vier Einschub-Server konfiguriert. Pro Server sind fünf 7,68 TB NVMe-SSD-Laufwerke als Massenspeicher und zwei Mellanox 100 Gb/s Netzwerkkarten für die Konnektivität verbaut. Genauere Details der Spezifikationen der NVMe-Server sind der Tabelle 2.1 zu entnehmen.

| Komponente | Konfiguration | Bemerkung |
|---|---|---|
| Prozessor | 2 x Intel(R) Xeon(R) Silver 4112 CPU @ 2.60 GHz | 4 Kerne, 8 Threads, 8,25 MB L3 Cache, 64-Bit Architektur |
| Mainboard | Supermicro X11DPT-B | Dual Socket P (LGA-3647), Intel® C621 Chipset |
| Arbeitsspeicher | 128 GiB | DDR4 2933MHz, ECC |
| Input und Output | (extern) 2 x USB, IPMI LAN, VGA, (intern) 2 x PCIe 3.0 x16, SIOM Slot, PCIe 3.0 x24 | SIOM: Netzwerk Add-on Card, Supermicro storage add-on card (PCIe x24) |
| Laufwerke | 6 x Hot-swap 2,5" NVMe-Schächte (PCIe 3.0 x24), 2 x M.2 NVMe/SATA-Stecker | 5 x 2,5" PM983 MZQLB7T6HMLA-00007, 2 x M.2 KXG60ZNV256G |
| Erweiterungskarten | AOC-MGP-i2 (SIOM), MCX516A-CCAT (PCIe x16) | 2 x 1 Gbps RJ45 (i350), 2 x 100 Gbps QSFP28 (Mellanox MT27800 - ConnectX-5) |
| Management | ASPEED AST2500 BMC | IPMI 2.0 + KVM, Monitoring für CPU, Chipset-Spannung, Speicher... |
| Netzteil | ein redundantes Netzteil für alle Nodes | 80 Plus Titanium Zertifiziert |

Tabelle 2.1: Hardware-Spezifikationen der Nodes [11]

Dem Datenblatt [16] der NVMe-Laufwerke können die Spezifikationen wie folgt entnommen werden:

| Eigenschaft | Spezifikation |
|---|---|
| verwendbare Kapazität | 7,68 TB |
| Schnittstelle | PCIe Gen 3 x4 @ 32 Gb/s |
| Sequential Read (128 KB) | maximal 3100 MB/s |
| Sequential Write (128 KB | maximal 2000 MB/s |
| Random Read (4 KB, QD32) | maximal 540.000 IOPS |
| Random Write (4 KB, QD32) | maximal 50.000 IOPS |

Tabelle 2.2: Technische Daten Samsung PM983

Entsprechend den Daten aus den Tabellen 2.1 und 2.2 beträgt die für die Hardwarekonfiguration akkumulierte Lesegeschwindigkeit eines NVMe-Servers mit fünf Laufwerken 15500 MB/s bzw. 124 Gb/s. Demnach ist die theoretische gebündelte Übertragungsrate von 200 Gb/s der Mellanox Netzwerkadapter in der Konfiguration adäquat[1].

### Netzwerkverbindung

Die 1 Gb/s-Schnittstellen und die IPMI-Netzwerk-Schnittstelle sind per Ethernet mit dem Netzwerk des Rechenzentrums und dem Internet verbunden. Über diese Schnittstelle erfolgen die Konfiguration, Wartung und das Management der NVMe-Server.

Die NVMe-Server sind über die Dual-Port 100 Gb/s Ethernet Adapter (Mellanox) mit einem 100 Gb/s-Switch verbunden. Die GPU-Server, die ebenso mit Mellanox 100 Gb/s Netzwerkkarten konfiguriert wurden, sind ebenso mit dem Switch verbunden. Die 100 Gb/s-Netzwerkadapter werden manuell konfiguriert und sind nicht mit den bestehenden Netzwerkkomponenten des Rechenzentrums verbunden.

Die Mellanox MT27800-Adapter unterstützen den Ethernet-Standard, sowie Remote direct memory access (RDMA) und RDMA over Converged Ethernet (RoCE).

RDMA ist ein Feature, welches ohne Umweg über die CPU, die direkte Kommunikation von Systemkomponenten über eine entfernte Verbindung ermöglicht. In dem 100 Gb/s-Ethernet-Netzwerk wird zwischen den verschiedenen Servern das *RDMA over Converged Ethernet*-Protokoll verwendet. Das Protokoll ermöglicht hohen Durchsatz, niedrige Latenzen und bietet eine hohe Zuverlässigkeit (vgl. [10]). Auch die Grafikkarten der GPU-Server unterstützen die *GPUDirect Storage*-Technologie (vgl. [15]) für den direkten Zugriff (“RDMA”) zwischen Grafikkarte und den Mellanox-Adaptern bzw. dem Storage-Dienst des verteilten Dateisystems.

[1] Die Schnittstelle der Netzwerkkarte beschränkt die Bandbreite mit ca. 126 Gb/s (MCX516A-CCAT Schnittstelle: PCIe 3.0 x16 [14]).

Mit dem in Abbildung 2.1 beschriebenen Aufbau der Komponenten wird erreicht, dass die Kommunikation zwischen NVMe-Servern und GPU-Servern möglichst niedrige Latenzen einhalten und der Durchsatz nicht durch die Bandbreite der Netzwerkkomponenten eingeschränkt wird. Die restlichen Systemkomponenten (NFS-Server, Netzwerk) werden außerdem entlastet.

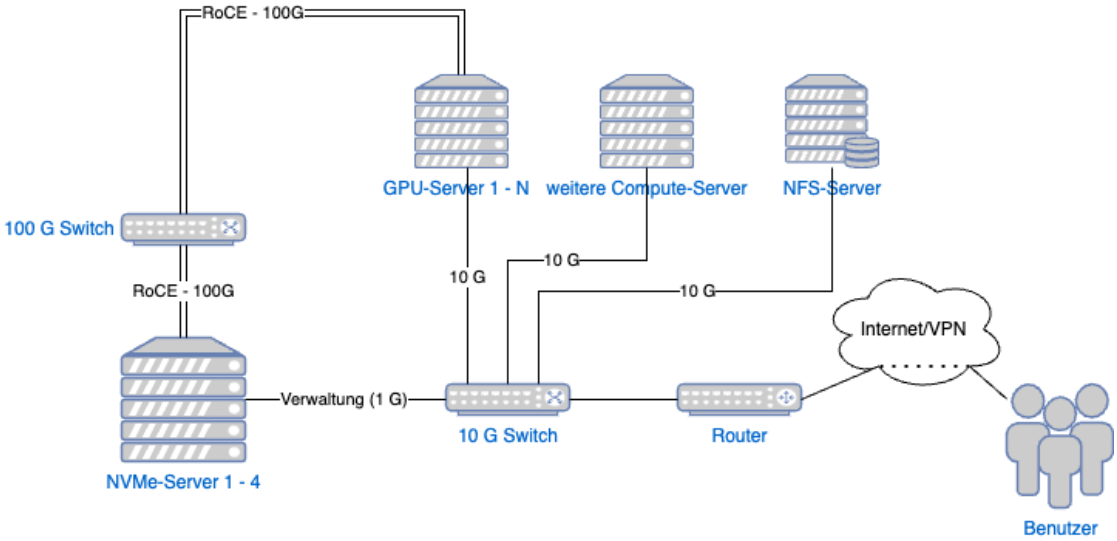


Abbildung 2.1: Technischer Aufbau des Rechenzentrums mit NVMe-Server als verteiltes Dateisystem
Quelle: Eigene Darstellung

## 2.2 Verteilte Dateisysteme

Um besser zu verstehen, was genau die Herausforderungen verteilter Dateisysteme sind, werden zunächst wichtige Funktionen eines lokalen Dateisystems erklärt.

Grundsätzlich gilt: Anwendungen, die auf einem normalen Computer-Betriebssystem ausgeführt werden, greifen auf Dateien und Verzeichnisse zu. Verzeichnisse dienen zur Strukturierung der Daten.

Als Dateisystem wird im Allgemeinen die Methode beschrieben, die das Betriebssystem verwendet, um den Datenzugriff auf Datenträgern umzusetzen. Der Dateisystemtyp sowie die technischen Eigenschaften der Datenträger beeinflussen die Leistung des Dateisystems.

Dateien bestehen aus Daten, die den eigentlichen Inhalt der Dateien enthalten. Die Größe der Daten variiert. Damit das Betriebssystem die Daten auf dem Datenträger finden kann, wird in UNIX-Dateisystemen eine Liste mit Inodes geführt, die Position und Metadaten jeder Datei enthält. Der genaue Aufbau unterscheidet sich je nach Dateisystem.

Als Beispiel für ein klassisches UNIX-Dateisystem wird beim ext2-Dateisystem der Speicher in Blockgruppen aufgeteilt. Inodes enthalten die Metadaten und verweisen auf die Datenblöcke (vgl. Abbildung 2.2). Metadaten enthalten Informationen zur Zugriffsberechtigung, Eigentümerschaft, Dateigröße und Weiteres. Der benötigte Speicher entspricht im Gegensatz zu den Daten immer einer festen Größe. Beim ext2-Dateisystem beträgt die Größe eines Inode 128 Bytes (vgl. [26]). Selbst eine Datei ohne Inhalt nimmt auf dem Laufwerk also mindestens so viel Platz ein wie die Größe der Metadaten.

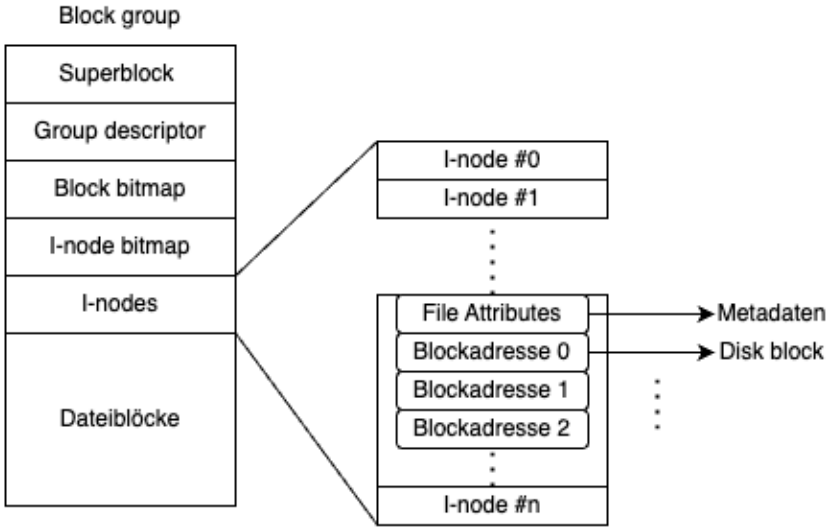


Abbildung 2.2: Aufbau eines Dateiblocks und Inodes im ext2-Dateisystem
Quelle: Eigene Darstellung in Anlehnung an [41]

Wie in Abschnitt 1.1 beschrieben, gibt es Anwendungsfälle, die besonders hohe Ansprüche an die Leistung des Dateisystems haben. Dateisysteme stellen grundsätzlich auf einem lokalen System Funktionen bereit. Für Anwendungsfälle in Rechnernetzen wurden verteilte Dateisysteme entwickelt, die es ermöglichen, über das Netzwerk auf Dateisysteme anderer Computer zuzugreifen [20].

Ein **verteiltes Dateisystem** ist für verteilte Anwendungen, die mit Daten arbeiten, oftmals eine der grundlegenden Komponenten (vgl. [42]).

Die Architektur des Zugriffs ist meist nach dem Client-Server-Modell konstruiert. Auch in dem Anwendungsfall dieser Arbeit ist die klassische Rollenverteilung sinnvoll. Dabei sind die NVMe-Server die **Server** und die GPU-Server die **Clients**.

Ein verteiltes Dateisystem bietet Zugriff auf ein entferntes Speicherlaufwerk für Dateien und Verzeichnisse. Ein Protokoll bestimmt dabei den Zugriff. Das Dateisystem kann von Clients eingehängt werden und es können mehrere Benutzer gleichzeitig auf eine Datei zugreifen.

Verteilte Dateisysteme als Server-Anwendung können auch weitere Funktionen wie Komprimierung der Daten, Replikation, erweiterte Zugriffsprotokolle und Backups beinhalten bzw. den Clients bereitstellen.

Generell können verteilte Dateisysteme auf einzelnen Computern ausgeführt werden.

Der Architektur eines Computers sind jedoch Grenzen gesetzt und die Erweiterung durch Hinzufügen weiterer Komponenten (vertikale Skalierung) ist beschränkt.

Speziellere verteilte Dateisysteme wurden entwickelt, um über mehrere Computer ein gebündeltes Dateisystem bereitzustellen. Solche Systeme werden als **clustered file system** oder **parallel file system** (paralleles Dateisystem) bezeichnet und bieten die Möglichkeit zur horizontalen Skalierung (Erweiterung durch Hinzufügen weiterer Computer) eines Dateisystems.

**Parallele Dateisysteme** sind besonders für den gleichzeitigen Zugriff von Prozessen auf Daten entwickelt (s. Abbildung 2.3). Diese Art von Dateisystemen wird oft in Hochleistungsrechenumgebungen bzw. Cluster-Computing-Systemen eingesetzt. Parallele Dateisysteme zeichnet aus, dass die Daten auf mehreren Servern (Nodes) verteilt gespeichert werden. Die Kommunikation der Server untereinander erfolgt über das Netzwerk. Dabei entsteht eine zusätzliche Latenz gegenüber lokaler Datenträger oder verteilter Dateisysteme bestehend aus einem Server. Die Herausforderung solcher Systeme ist die bei der Kommunikation zwischen den Komponenten entstehende Latenz.

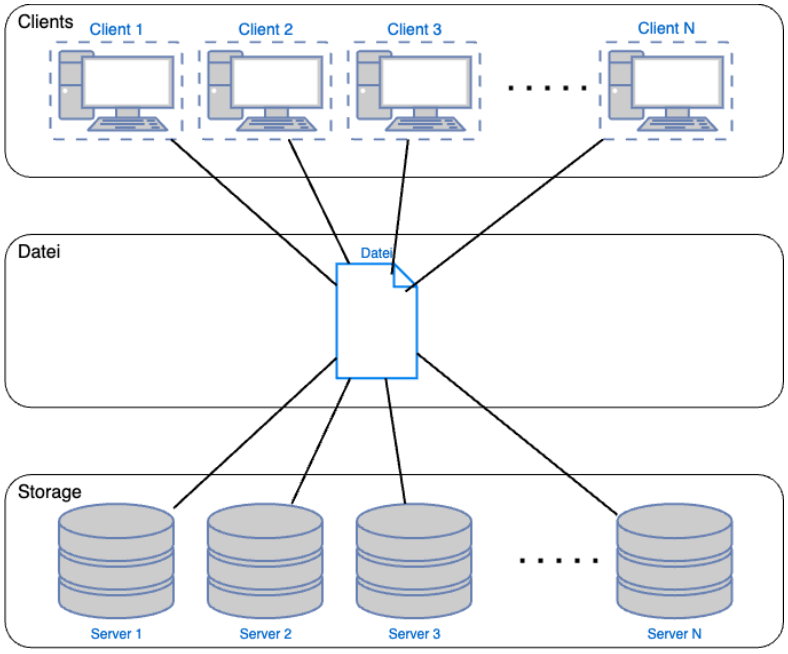


Abbildung 2.3: Paralleler Zugriff mehrerer Clients auf eine Datei, die auf mehreren Speichergeräten verteilt abgelegt ist
Quelle: Eigene Darstellung in Anlehnung an [29]

In der Konstruktion des Systems können dedizierte Server für das Speichern der Metadaten (Metadaten Server) und Daten (Storage Server) gewählt werden. Storage Server sind meist mit mehreren großen Laufwerken konfiguriert. Als Datenträger für Metadaten Server sollten schnelle Laufwerke mit möglichst geringer Latenz verwendet werden (bspw. NVMe-Laufwerke). Auch ein Aufbau, bei dem Metadaten Service und Storage Service auf einem Server parallel ausgeführt werden, ist mit vielen parallelen Dateisystemen möglich.

Für das Anwendungsszenario dieser Arbeit ist die Hardware mit gleicher Konfiguration der Laufwerke pro Server vorgegeben und entspricht am ehesten einer solchen symmetrischen Konfiguration, bei der die Server parallel als Metadaten Server und Storage Server betrieben werden.

## 2.3 Anforderungsanalyse und Wahl eines verteilten Dateisystems

Dieser Abschnitt befasst sich mit der Analyse der Anforderungen, die an das verteilte Dateisystem gestellt werden.

Wie in der Einleitung beschrieben wurde, soll ein verteiltes Dateisystem für die Installation ausgewählt werden, welches horizontal skaliert. Der Fokus der Anforderungen liegt auf niedriger Latenz, hohem Datendurchsatz und der einfachen Installation und Wartung mit Automatisierungsskripten.

Im Folgenden werden wichtige Anforderungen (nicht-funktionale Eigenschaften) verteilter Dateisysteme beschrieben und deren Relevanz für die in Abschnitt 1.3 beschriebenen Ziele wird anschließend bewertet. Die Tabelle 2.3 zeigt die Eigenschaften eines verteilten Systems mit Fokus auf *verteilte Dateisysteme* (vgl. Kapitel 11 aus [42]). Anschließend werden die Anforderungen, entsprechend der Bedürfnisse des Anwendungsfalls, genauer beschrieben. Am Ende des Abschnitts werden, entsprechend der Anforderungen, passende Systeme vorgestellt.

| Eigenschaft | Kurzbeschreibung |
|---|---|
| Skalierbarkeit | Skalierung beschreibt die Vergrößerung des Systems in Hinsicht auf Systemressourcen, Benutzeranzahl und geografische Standorte. |
| Synchronisierung | Die Herausforderung des gleichzeitigen Zugriffs mehrerer Benutzer auf Dateien. |
| Replikation und Konsistenz | Replikation der Daten durch gespiegelte Server oder clientseitiges Caching. |
| Fehlertoleranz | Verfügbarkeit, Zuverlässigkeit, Funktionssicherheit und Wartbarkeit. |
| Sicherheit | Authentifizierung und Zugriffssteuerung. |
| Leistungseffizienz | Overhead und Effizienz des Systems. |
| Verwaltbarkeit | Einfache Installation, Monitoring, Erweiterung. |
| Versionierung/Wiederherstellung | Versionsverwaltung, Zurückspringen an einen alten Zustand der Dateien. |

Tabelle 2.3: Wichtige Anforderungen verteilter Dateisysteme (vgl. [42])

### Beschreibung der Anforderungen mit Fokus auf die Zielsetzung

- **Skalierbarkeit** – Das System soll skalierbar sein in Hinsicht auf die Leistung (Durchsatz und Anzahl gleichzeitiger Zugriffe) und verfügbare Dateisystemkapazität. Mit der Skalierung des Systems steigt potenziell die Latenz des Systems. Die Latenz des Systems sollte durch eine schnelle Netzwerkinfrastruktur möglichst minimal sein. Das räumliche bzw. geografische Wachsen des Systems begrenzt sich momentan auf ein Rechenzentrum an einem Standort. Eine Verbindung mehrerer Standorte und des verteilten Dateisystems ist für diesen Anwendungsfall nicht vorgesehen.

- **Synchronisierung** bzw. Sicherheitsmechanismen für den Zugriff – Der Zugriff auf das Dateisystem sollte nicht ungeschützt sein. Der Dateisystemzugriff soll durch den Administrator vorgegeben sein und kein direkter Zugriff durch Benutzer auf das Dateisystem soll erfolgen. Synchronisierungsprobleme durch gleichzeitigen Zugriff auf Dateien sollen durch POSIX I/O-Konformität (vgl. [1]) gewährleistet sein.

- **Replikation und Konsistenz** – Falls Systemkomponenten bzw. ganze Server ausfallen, sollen Replikationsmechanismen das System unterbrechungsfrei online halten. Aufgrund der aktuellen Größe des Rechenzentrums und des Anwendungsgebiets als Zwischenspeicher wird in der Umsetzung auf die Ausfallsicherheit mit Spiegelservern gegenüber verminderter Leistung verzichtet. Die Möglichkeit einer Konfiguration mit Spiegelservern ist trotzdem wünschenswert, falls das System vergrößert wird. Der gleichzeitige Zugriff beschränkt sich momentan auf die Anzahl der vorhandenen Compute-Server. Das Caching der Daten clientseitig soll entweder durch den Linux Page Cache (vgl. [33]) durch den Kernel oder auf Anwendungsebene realisiert sein.

- **Fehlertoleranz** – Ausfallsicherheit wird durch Replikationsmechanismen erzielt. Dem Hardwareausfall eines Datenträgers kann mit Konfiguration eines RAID-Systems entgegengewirkt werden. Spiegelserver sollen bei einem Komplettausfall eines Servers das System aufrechterhalten. Eine automatische Wiederaufnahme des Betriebs nach Ausfall einer Komponente gehört zu wünschenswerten Funktionen.

- **Sicherheit** – In Client-Server Systemen ist meist der Server für die Zugriffssteuerung verantwortlich. Im Anwendungsfall dieser Arbeit soll der Zugriff auf eine limitierte Anzahl von Systemen beschränkt sein und der direkte Zugriff durch Benutzer ist nicht vorgesehen.

- **Leistungseffizienz** – Das System sollte auf optimale Lese- und Schreibleistung ausgelegt sein. Eine hohe Geschwindigkeit beim Schreiben vieler kleiner Datenblöcke ist eine Herausforderung von Dateisystemen und insbesondere auch verteilter Dateisysteme [54]. Geringe Systemlast bzw. Overhead durch das lokale Dateisystem oder Verwaltungsprozesse sind wünschenswert. Dazu gehört auch die Unterstützung für Funktionen wie RDMA zur Minimierung der Latenz.
- **Verwaltbarkeit** – Die Verwaltbarkeit ist ein Aspekt der Skalierbarkeit. Ein kleines Team sollte möglichst einfach das System verwalten können und der Verwaltungsaufwand sollte minimal sein. Die Installation des Systems sollte möglichst einfach und mit Skripten automatisiert werden können. Ein minimales Monitoring des Systems sollte gegeben sein. Es kann auch durch externe Systeme eine Hardwareüberwachung erfolgen.
- **Versionierung und Wiederherstellung** – Die Versionierung von Daten ist für den Anwendungsfall als Zwischenspeicher nicht erforderlich. Anwenderfehler in Form von versehentlichem Löschen sind zu vernachlässigen.

### Auswahl des Systems

Entsprechend der vorangegangenen Beschreibung gibt es einige verteilte Dateisysteme, die für diese Art der Anwendung zur Auswahl stehen und in Hochleistungsrechenzentren eingesetzt werden.

In der TOP500-Liste (vgl. [35]) von Hochleistungsrechenzentren wird ausschließlich Linux als Betriebssystem verwendet, so auch im Rechenzentrum des FTZ SMSY. Bei der Wahl des Systems wird ein POSIX-konformes Dateisystem vorausgesetzt, wie es unter Linux fast immer der Fall ist [29, 35].

Verbreitet und in der TOP500-Liste vertreten sind zum Beispiel die Systeme **Lustre**, **Spectrum Scale** und **BeeGFS** (vgl. [29, 55, 7, 8]).

Die folgende Tabelle zeigt die Eigenschaften der Systeme entsprechend der Anforderungen. Die Eigenschaften wurden, so weit es möglich ist, aus der Dokumentation und von den Herstellerseiten entnommen. Weitere Eigenschaften beschreiben die unterstützten Schnittstellen, Features und das Lizenzmodell.

| Eigenschaft | Beschreibung |
|---|---|
| Skalierbarkeit | Alle Systeme sind skalierbar bzw. bieten die Möglichkeit mehrere Server zu verbinden. Ebenso wird unterstützt mehrere Dienste (Daten und Metadaten) auf einem Server auszuführen (Converged Setup). Lustre unterstützt nicht die gleichzeitige Ausführung von Server-Komponenten und Client auf einem System.<br>**Lustre** - Maximale Dateigröße 32 Petabyte (PB) bei Lustre-Dateisystem (basierend auf ext4), 16 Exabyte (16 EB entspricht 16.000 PB) bei ZFS. Maximale Dateisystemgröße 16 EB.<br>**BeeGFS** - Maximale Dateigröße 16 EB. Maximale Dateisystemgröße 16 EB.<br>**Spectrum Scale** - Maximale Dateigröße 8 EB. Maximale Dateisystemgröße 8 Yottabyte (YB) (entspricht 8.000.000 EB). |
| Synchronisierung | Alle Systeme sind POSIX-konform. Eine Zugriffssteuerung ist gegeben (siehe *Sicherheit*). |
| Replikation und Konsistenz | Alle Systeme bieten Funktionen zur Daten- bzw. Serverreplikation die je nach System manuell konfiguriert werden müssen oder automatisch aktiv sind. |
| Fehlertoleranz | Alle Systeme bieten durch Replikationsmechanismen Ausfallsicherheit und damit mindestens einen Grad an Fehlertoleranz. |
| Sicherheit | Alle Systeme unterstützen externe Authentifizierungsdienste.<br>**Lustre** und **BeeGFS** bieten eine einfache Möglichkeit für die Authentifizierung von Server-Komponenten und Clients per Shared Secret. |
| Leistungseffizienz | Alle Systeme versprechen hohe Leistung und Effizienz bei niedrigen Hardware-Anforderungen. Die Komponenten unterstützen teilweise manuell konfigurierbare Anzahl an Threads.<br>**Lustre** empfiehlt für Server mindestens 64 GB Arbeitsspeicher.<br>Eine **Spectrum Scale** Installation setzt mindestens vier Server voraus und empfiehlt für gute Leistung Server mit mindestens 64 GB Arbeitsspeicher und Prozessoren mit mindestens 24 Kernen. |

| | |
|---|---|
| Verwaltbarkeit | Alle Systeme bieten eine Funktion zur Gruppierung der Storage-Kapazität in verschiedene Leistungsabstufungen (Storage Tiering).<br>Die Installation von **Lustre** und **BeeGFS** erfolgt über Befehle in der Kommandozeile.<br>Die Installation der Komponenten von **Spectrum Scale** ist manuell, mit einem Toolkit oder per GUI möglich. |
| Versionierung und Wiederherstellung | **Spectrum Scale** unterstützt eingebaute Funktionen für Backup und Snapshot. |
| Schnittstellen | Alle Systeme bieten ein POSIX-konformes Client Modul.<br>**Lustre** ermöglicht zusätzlich den Zugriff per NFS und SMB.<br>**Spectrum Scale** bietet Unterstützung für NFS, Samba, Object und HDFS connector (Hadoop kompatibel). |
| Features und weiteres | RDMA und GPUDirect-Storage werden von allen Systemen unterstützt.<br>**Lustre** unterstützt offiziell RHEL ab Version 7 und Derivate.<br>Von **BeeGFS** unterstützte Betriebssysteme sind RHEL (Alma-Linux, Rocky Linux), SLES, Debian und Ubuntu. Bei Ausfall von RDMA erfolgt der automatische Wechsel zu einer TCP/IP-Verbindung.<br>Von **Spectrum Scale** unterstützte Betriebssysteme sind Red Hat Enterprise, SLES und Ubuntu. Es wird zusätzlich NVIDIA SuperPOD unterstützt. |
| Linzenzierung | **Lustre** ist ein reines Open Source Projekt (GPL v2, LGPL).<br>**BeeGFS** ist frei erhältlich für nichtkommerzielle Verwendung. Source Code ist einsehbar. Client ist GPL v2, Serverkomponenten BeeGFS End User License Agreement [44].<br>**Spectrum Scale** umfasst drei verschiedene Lizenzabkommen: IBM Spectrum Scale Client license, IBM Spectrum Scale server license und IBM Spectrum Scale FPO license (vgl. [13]). Eine kostenlose Testversion kann mit einem Dateisystem bis 12 TB betrieben werden, sonst monatliches oder dauerhaftes Lizenzmodell pro Terabyte. |

Tabelle 2.4: Systemeigenschaften

Alle drei Systeme erfüllen grundsätzlich die Anforderungen.

Die Einarbeitung in die Dokumentation[2] der verschiedenen Systeme und die Gegenüberstellung der Eigenschaften in Tabelle 2.4 haben darüber entschieden, welches System in der Arbeit näher betrachtet und installiert wird.

Es wurde entschieden BeeGFS zu installieren, da die gleichzeitige Ausführung von Server-Komponenten und Client unterstützt wird, die für die Evaluierung vorausgesetzt wird. Ebenso bietet BeeGFS eine sehr übersichtliche Dokumentation des Systems, die für die Verwaltung und Wartbarkeit von großem Vorteil sind.

Lustre ist für die Installation ausgeschieden, da die parallele Ausführung der Server-Komponenten und Client nicht möglich ist, die für die Evaluation erforderlich ist.

Spectrum Scale wurde wegen zu großer Komplexität und Lizenzierung nicht für eine Testinstallation gewählt.

## 2.4 BeeGFS

In diesem Abschnitt wird die Architektur des parallelen Dateisystems BeeGFS genauer vorgestellt. Es werden die Funktionen und Komponenten erläutert und anschließend die Konfigurationsmöglichkeiten für das grundlegende Dateisystem und weitere Funktionen eruiert.

BeeGFS ist ein paralleles Dateisystem, entwickelt vom Fraunhofer Bereich für High Performance Computing, für High Performance Computing Anwendungen mit Fokus auf Leistung.

Wie in den Proceedings von Chowdhury et al. (2019) [8] und Boito et al. (2022) [4] untersucht, ist BeeGFS ein aktuelles und verbreitetes paralleles Dateisystem, welches in Rechenzentren mit Anwendungen aus dem Bereich Deep Learning verwendet wird.

[2] Dokumentationen der Systeme:
Lustre - https://www.lustre.org/documentation/
BeeGFS - https://doc.beegfs.io/
Spectrum Scale - https://www.ibm.com/docs/en/storage-scale

### 2.4.1 Architektur

In einem parallelen Dateisystem werden die Aufgaben eines lokalen Dateisystems in separate Dienste aufgeteilt.

BeeGFS ist in die Server-Dienste *Management*, *Storage* und *Metadata* aufgeteilt. Die Aufteilung der Dienste in Komponenten entspricht der klassischen Architektur eines parallelen Dateisystems (vgl. Abbildung 2.4). Es können mehrere Instanzen von Storage- und Metadata-Komponente ausgeführt werden.

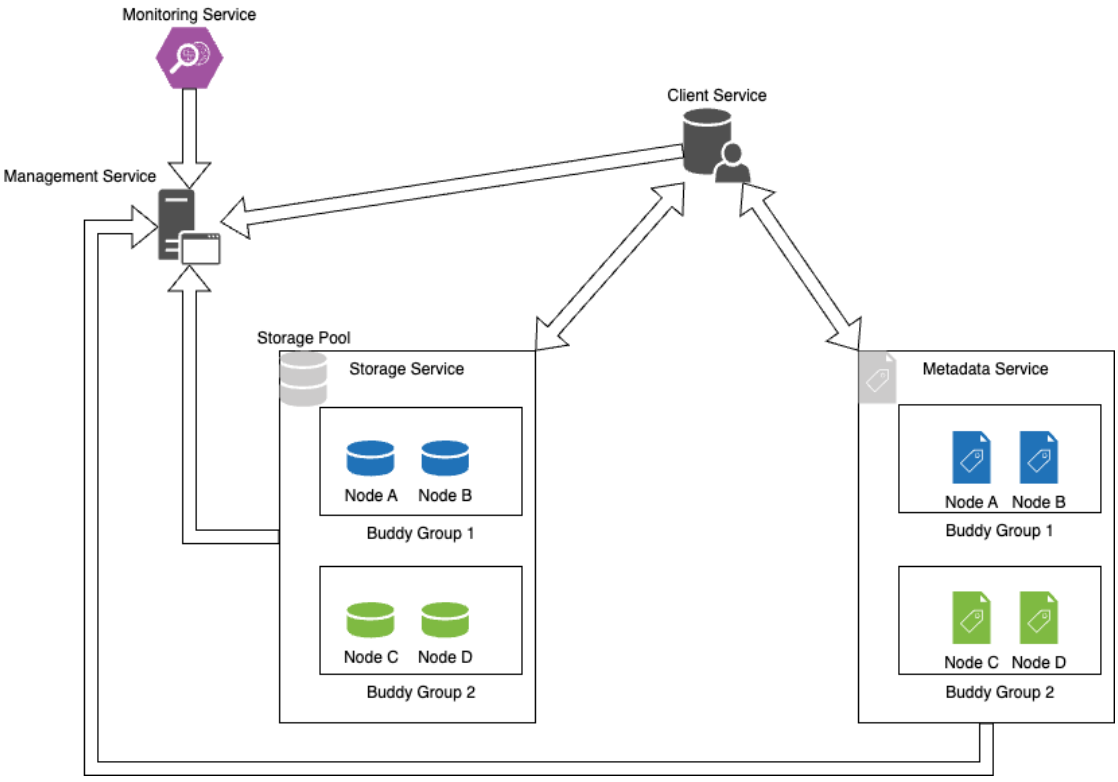


Abbildung 2.4: Architektur von BeeGFS
Quelle: Eigene Darstellung in Anlehnung an [43]

Daten werden nach dem Stripe-Prinzip (vgl. Abbildung 2.5) auf mehrere Storage Server verteilt und dadurch die Last auf alle Server verteilt.

Der Zugriff von Clients erfolgt parallel auf mehrere Storage Server, auf denen die Daten abgelegt sind, und auch wenige oder ein einzelner Client mit schneller Anbindung kann von dem parallelen Dateisystem gegenüber einem einzelnen Dateiserver profitieren.

Die Leistung des Metadata Server ist ausschlaggebend für eine möglichst geringe Latenz beim Zugriff der Clients. Mit steigender Anzahl an Clients profitiert das System von mehreren Metadata Servern. Beispielsweise, wenn mit einem `ls`-Befehl in einem Verzeichnis viele Dateien in Unterverzeichnissen aufgelistet werden, melden mehrere Metadata Server einen Teil des Ergebnisses zurück.

BeeGFS bietet die Möglichkeit Dienste auf einem Server mehrfach auszuführen. Es kann beispielsweise in einer heterogenen Server-Landschaft mit unterschiedlicher Anzahl von Laufwerken gewünscht sein, pro Laufwerk eine Instanz des Storage Services auf dem Server auszuführen.

Wie in Abschnitt 2.1 beschrieben, soll die Verbindung der Server-Komponenten oder Client per RDMA-Technologie erfolgen, um niedrige Latenzen zu einzuhalten. Sofern der Netzwerkkartentreiber die Funktionalität bietet, ist die Verbindung per RDMA standardmäßig möglich.

Im Folgenden werden die Eigenschaften der grundlegenden Dienste erläutert.

#### Management Service

Die Management-Komponente übernimmt die Zugriffskontrolle. Der Management Service agiert als Wurzel-Server für weitere Komponenten. Jede weitere Server-Komponente und jeder Client registriert sich beim Management Service. Die Systemvoraussetzungen sind niedrig und der Dienst benötigt für die Daten kein schnelles Laufwerk oder viel Speicherplatz. Der Dienst ist bei Datenoperationen nicht involviert und beeinflusst nicht die Leistung des Dateisystems.

Das Registrieren beim Managment Service kann mit einem *Shared Secret* geschützt werden. Ein Shared Secret ist ein Passwort oder eine Schlüsseldatei, die mit den Instanzen geteilt ist, um sich bei der Kommunikation zu authentifizieren. Ist das System installiert, konfiguriert und alle Server wurden beim Management Service registriert, kann das Hinzufügen weiterer Server gesperrt werden.

Der Management Service agiert auch als Watchdog (Überwachungsdienst). Die Ausführung des Management Service kann auf einem der Storage- oder Metadata-Nodes erfolgen. Alle weiteren Komponenten registrieren sich bei dem Management Service.

### Storage Service

Der Storage Service stellt den eigentlichen Speicherplatz für die Daten zur Verfügung. Es ist üblich, dass der Storage Service auf mehreren Servern ausgeführt wird und mehrere Datenträger pro Server konfiguriert sind. Die Laufwerke werden „manuell“ formatiert und im Verbund oder als einzelne Ziellaufwerke (*Storage Targets*) eingehängt. Der Storage Service muss mit mindestens einem eingehängten Laufwerk als Storage Target konfiguriert werden. Die Daten, die ein Client schreiben möchte, werden vom Storage Service in Stripes aufgeteilt und auf einem *Storage Pool* (s. Abschnitt 2.4.2) gespeichert. Die Verteilungsfunktion für die Stripes der Daten erfolgt nach dem Round-Robin-Ansatz (vgl. Abbildung 2.5).

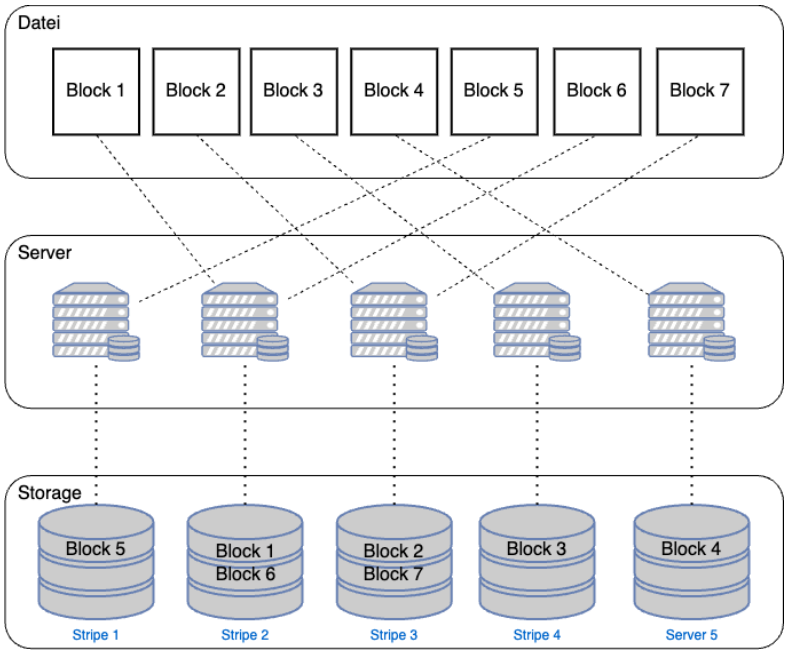


Abbildung 2.5: Aufteilen der Blöcke (Stripes) auf mehrere Server (Striping) nach dem Round-Robin-Ansatz
Quelle: Eigene Darstellung in Anlehnung an [29]

Die Datenträger aller Storage Server werden im Verbund zusammengefasst, und dadurch werden die Kapazität und Leistung aggregiert. Je nach Konfiguration werden durch Techniken wie Spiegelung (Mirroring) auf mehreren Servern Duplikate der Daten erstellt, um Ausfallsicherheit zu erzielen.

### Metadata Service

Auf Datenträgern des Metdata Service werden die Metadaten-Informationen zu den auf den Storage Servern abgelegten Daten gespeichert. Mehrere Metadata Server können konfiguriert werden, um die Latenz bei vielen Zugriffen gering zu halten. Der Zugriff auf Metadaten ist Grundlage für die Lese- und Schreibvorgänge. Der Dienst koordiniert das Striping der Daten, entscheidet also, welche Datenblöcke auf welchen Storage Server abgelegt werden, und informiert den Client über den Speicherort vorhandener Dateien.

Ein Metadata Server enthält immer Metadaten ganzer Verzeichnisse. Die Aufgaben sind Erstellung, Löschung und Abfrage von Metadaten entsprechend aller Dateien und Verzeichnisse die auf dem Dateisystem gespeichert werden.

Die Leistung des Metadata Servers hängt stark von dem grundlegenden Dateisystem ab. Um niedrige Latenzen im System einzuhalten, sollten die Metadata Server mit Laufwerken mit geringer Latenz konfiguriert werden.

### Client

Der Client ist als Linux Kernel Modul implementiert und erhält über einen Einhängepunkt im Dateisystem Zugriff auf das verteilte Dateisystem.

Der Client erhält beim Lese- oder Schreibzugriff vom Metadata Service die Information, von welchem Storage Server Datenblöcke geladen oder auf welchem Storage Server Datenblöcke gespeichert werden können. Der Client greift direkt auf die Storage Server zu.

Der *Caching Mode* der Clients kann als `buffered` (Standard) oder `native` konfiguriert werden. Die Konfiguration `native` verwendet den Linux kernel page-cache (vgl. [33]) und kann, entsprechend vorhandenem Arbeitsspeicher, größere Mengen zwischenspeichern. Wenn der Client gleichzeitig mit Server-Komponenten ausgeführt wird (wie in der Evaluation), sollte die Option `buffered` (kleiner Buffer für write-back und read-ahead) verwendet werden (vgl. [46]), damit die Leistung der Server-Komponenten nicht beeinflusst wird.

### 2.4.2 weitere Dienste und Features

Nachfolgend werden zusätzliche Dienste und Funktionen erklärt. Diese sind für die grundlegende Funktion nicht notwendig und nur für spezifischere Konfiguration wichtig.

#### Monitoring

Ein optionaler *Monitoring Service* kann Leistungsdaten und Statistiken in einer Datenbank[3] sammeln. Mit einer Software zur Datenvisualisierung[4] können diese Daten in Graphen dargestellt werden [49].

Der Metadata Service und Client können konfiguriert werden, um Aktionen, die ein Client ausführt (zum Beispiel Datei lesen, Datei löschen, Datei verschieben) im JSON-Format weiterzugeben. Dabei können bestimmte Aktionstypen maskiert werden (vgl. [47]).

#### Storage Pools

Die Gesamtheit der in den Storage Services konfigurierten Storage Targets wird in *Storage Pools* gruppiert. Standardmäßig werden alle Storage Targets zu einem Storage Pool hinzugefügt. Es können bei Bedarf Storage Pools mit manuell zugewiesenen Storage Targets erstellt werden, um Datenträger-Gruppen in Leistungsklassen einzugrenzen. So können verschiedene Storage Pools zum Gruppieren von schnellen oder langsamen Laufwerken erstellt werden (*Storage Tiering)*. Beispielsweise kann ein Storage Pool mit schnellen Datenträgern für aktive Projekte konfiguriert werden und ein weiterer mit Laufwerken mit hoher Kapazität als Archiv.

#### Mirroring

Mirroring beschreibt die Replikation der Daten. Das Spiegeln zwei oder mehrer Server bzw. der Daten auf diesen wird eingesetzt, um hohe Verfügbarkeit und Ausfallsicherheit zu gewinnen.

In BeeGFS kann Replikation mit der Gruppierung von Servern in *buddy groups* konfiguriert werden. Diese verwalten untereinander die Datenreplikation. Dafür werden Storage

[3] Beispielsweise InfluxDB - https://www.influxdata.com/
[4] Beispielsweise Grafana - https://grafana.com/

Server und Metadata Server zu *buddy groups* hinzugefügt und als *Mirror Buddy Group* definiert.

### 2.4.3 Operationen

Im Folgenden wird der Ablauf bei Lese- und Schreiboperationen visualisiert.

#### Schreibvorgang

Möchte der Client eine neue Datei erstellen oder eine vorhandene Datei bearbeiten, wird der Metadata Service eine neue Datei hinzufügen oder den Speicherort der vorhandenen Datei zurückgeben. Dem Client wird mitgeteilt, auf welchen Storage Server die Datenblöcke geschrieben werden sollen, und eine direkte Verbindung zu den Storage Services wird aufgebaut.

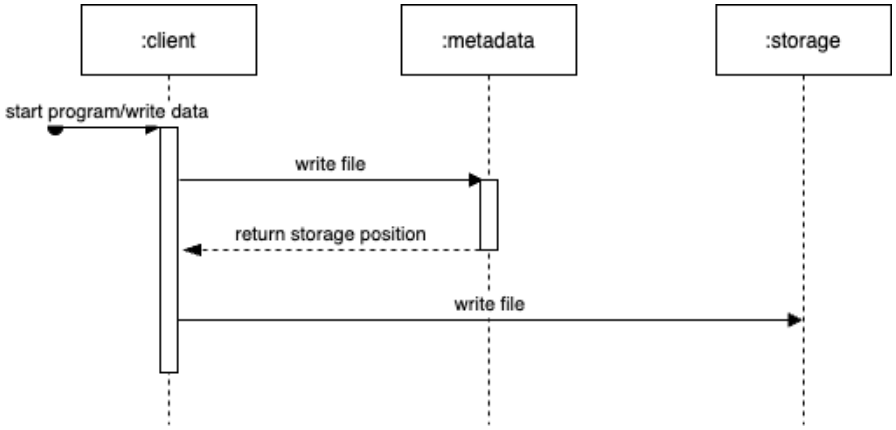


Abbildung 2.6: Schreiben einer neuen Datei/Bearbeiten einer Datei
Quelle: Eigene Darstellung

#### Lesen einer Datei

Möchte der Client eine Datei lesen, meldet der Metadata Service die Position der Datei zurück. Dem Client wird mitgeteilt, wo die Datenblöcke liegen, und eine direkte Verbindung zum Lesen von den Storage Services wird aufgebaut.

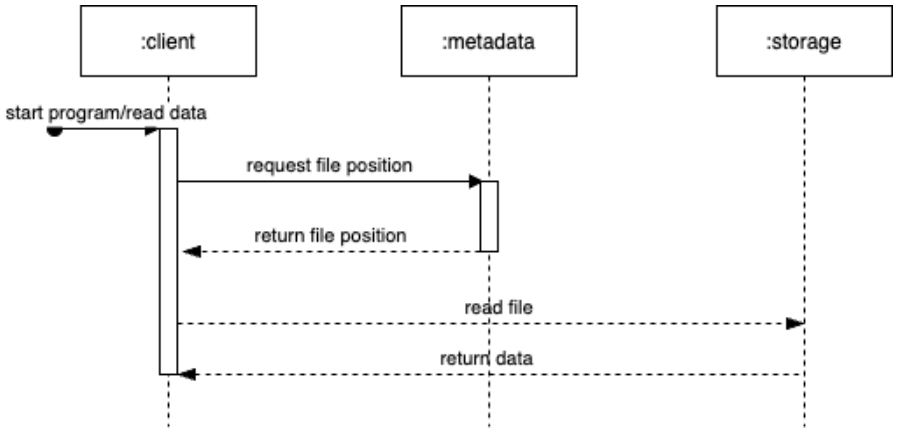


Abbildung 2.7: Lesen einer Datei
Quelle: Eigene Darstellung

### 2.4.4 Unterschiedliche Konfigurationsmöglichkeiten des Dateisystems

BeeGFS bietet keine integrierte Softwarelösung für ein Dateisystem (Block-Dateisystem). Die Konfiguration der grundlegenden Dateisysteme ist in verschiedenen Varianten möglich, die im Folgenden beschrieben werden.

Prinzipiell kann jedes POSIX-konforme Dateisystem als Grundlage für einzelne Laufwerke bzw. ein Software-RAID für mehrere Laufwerke im Verbund verwendet werden.

Wie in Abschnitt 1.3 beschrieben, ist es Teil dieser Arbeit zu evaluieren mit welcher Konfiguration die optimale Leistung für die Anwendung als Zwischenspeicher für die GPU-Server erzielt wird.

Die Konfigurationen unterscheiden sich durch verschiedene grundlegende Dateisysteme. In Abschnitt 4.1 werden die für die Evaluation gewählten Konfigurationen für das Dateisystem genauer beschrieben.

#### Multiple Targets

Eine Variante der Konfiguration ist, die einzelnen Laufwerke eines Servers zu formatieren und lokal einzuhängen. Für den Storage Service können dann mehrere Ziellaufwerke als Storage Targets konfiguriert werden. Das Striping der Datenblöcke auf die lokalen Laufwerke wird dann von dem Storage Service übernommen.

Für den Metadata Service wird pro Server ein Laufwerk verwendet.

#### Multi Mode

Im Vergleich zur Variante mit mehreren Ziellaufwerken für einen Storage Service, gibt es die Möglichkeit mehrere Storage Services pro Server zu konfigurieren und jeweils ein Laufwerk als Storage Target zu verwenden. Das Striping wird dann vom Metadata Service übernommen.

Ebenso wird für den Metadata Service pro Server ein Laufwerk verwendet.

#### Software-RAID

Mit **mdadm** oder **ZFS** kann ein Software-RAID Dateisystem erstellt werden. Dabei kann gewählt werden, ob vier der fünf Laufwerke als Software-RAID für den Storage Service verwendet werden oder ein Software-RAID mit allen fünf Laufwerken erstellt und für Storage Service und Metadata Service verwendet wird. Es kann das Dateisystem für Storage und Metadaten partitioniert werden oder es können beide Prozesse auf ein eingehängtes Dateisystem zugreifen.

## 2.5 Ansatz der Testmethodik - Parallele Benchmarks

Um die Leistung des parallelen Dateisystems zu Evaluieren, müssen entsprechend viele gleichzeitige Prozesse simuliert werden, die auf das System zugreifen.

BeeGFS bietet in der Dokumentation (vgl. [45]) Hinweise, wie parallele Benchmarks ausgeführt werden können.

Da in der Testumgebung keine weiteren Server als Client-Systeme zur Verfügung stehen, werden die Benchmarks parallel auf den Servern ausgeführt. Dazu wird auf den Servern das Client Kernel Modul gestartet und das BeeGFS-Verzeichnis im lokalen Dateisystem eingehängt.

Zum Testen der Leistung des Dateisystems wird das Tool **IOR** [30] verwendet, um die Durchsatz-Leistung zu messen.

Dabei werden verschiedene Anwendungsfälle betrachtet:

**Multi-stream throughput:** In diesem Testszenario ist die Blockgröße an die Chunksize[5] des Dateisystems angeglichen, also gleich oder ein Vielfaches der Chunksize. Dies entspricht einem optimalen Szenario.

**Shared file throughput:** Die Blockgröße ist ungleich der Chunksize des Dateisystems. Dies entspricht einem realistischen Testszenario mit unterschiedlichen Dateigrößen entsprechend einem realen Dataset.

Mithilfe des Tools **MDTest** [30] wird die Leistung der Metadaten-Komponente gemessen. Der Test erstellt, löscht und listet den Status von Dateien mit einer Größe von 0 Byte und misst, wie viele dieser Vorgänge (Operationen) pro Sekunde auf dem Dateisystem ausgeführt werden können. Input/Output operations Per Second (IOPS) ist eine Leistungsangabe, die beschreibt, wie viele Ein- und Ausgabe-Befehle pro Sekunde ausgeführt werden können. Die Zugriffslatenz kann davon abgeleitet werden. Es werden bei den Ergebnissen die entsprechenden Werte *file creations/s*, *file stats/s* und *file removal/s* betrachtet.

**IOPS:** Bei diesem Testszenario werden sehr kleine Dateien (4 KiB) geschrieben. Dieses Testszenario nähert sich einem Metadaten-Benchmark an.

[5] Chunksize entspricht der Größe der Stripes. Die Größe kann manuell konfiguriert werden. (Default: 512 KB)

Beide Anwendungen basieren auf dem Message Passing Interface-Standard (**MPI**). Mit Hilfe des MPI können mehrere Instanzen der Benchmark-Tools parallel auf den Servern ausgeführt werden. In diesem Fall muss sichergestellt sein, dass auf allen Servern, auf denen der Benchmark laufen soll, das Dateisystem unter dem gleichen Verzeichnis eingebunden ist (gleiche Client-Konfiguration) und die Firewall die Kommunikation nicht blockiert.

Im Abschnitt 3.3 wird genauer erläutert, mit welchen Parametern die Benchmarks ausgeführt werden.

# 3 Umsetzung

In diesem Kapitel wird das Vorgehen bei der Installation von BeeGFS beschrieben. In Abschnitt 3.2 wird geschildert, wie die automatisierte Installation umgesetzt wird. In Abschnitt 3.3 wird erläutert, wie genau die Benchmarks für die Evaluation des Systems ausgeführt werden.

## 3.1 Testinstallation

Für eine erste Testinstallation werden vier virtuelle Maschinen auf einem Server im Rechenzentrum verwendet. Auf den virtuellen Maschinen wird die grundlegende Installation des Systems erprobt, um sicherzustellen, dass das System wie erwartet funktioniert, eine problemlose Kommunikation im Netzwerk sichergestellt ist und das Betriebssystem kompatibel ist. Anschließend folgt die Installation auf der Bare-Metal Hardware. In Abschnitt 2.1 wurden die genaueren Details der Hardware beschrieben.

Als grundlegendes Betriebssystem für Server wird im Rechenzentrum des FTZ SMSY CentOS 8 verwendet. Um die Server-Landschaft homogen zu halten, ist wünschenswert, CentOS oder ein Derivat[1] für die Konfiguration zu verwenden.

### Installation auf virtuellen Maschinen

Das Betriebssystem der virtuellen Maschinen ist CentOS 8 mit Kernel 4.18.0-358. Die Installation von BeeGFS auf den virtuellen Maschinen erfolgt per Secure Shell (SSH) Zugriff von einem Client im VPN des Rechenzentrums. Dabei wird nach der Dokumentation [51] vorgegangen.

[1] Ein Derivat oder Fork ist ein (weiterer) Entwicklungszweig eines Linux-Betriebssystems, welches eine ähnliche Architektur zum Ursprung beibehält.

Eine virtuelle Maschine wird als Management Server gewählt. Auf allen weiteren virtuellen Maschinen sollen die Dienste für Metadata, Storage und Client installiert werden.

Ablauf:

1. Das Paket-Repository von BeeGFS wird heruntergeladen und hinzugefügt.
2. Auf den Servern werden die benötigten Komponenten per Paketmanager installiert.

   ```
   yum install beegfs-mgmtd beegfs-meta beegfs-storage beegfs-client
   ```

3. Als erste Komponente wird der Management Service konfiguriert. Dabei wird mit dem Parameter `p` das Verzeichnis angegeben, in dem der Management Service Daten speichert.

   ```
   /opt/beegfs/sbin/beegfs-setup-mgmtd -p /data/beegfs/beegfs_mgmtd
   ```

4. Nun können Metadata Service und Storage Service auf allen virtuellen Maschinen konfiguriert werden. Der Parameter `p` gibt das Verzeichnis an, in dem der jeweilige Service Daten ablegt, `s` gibt dem Service eine numerische ID und `m` entspricht dem Hostnamen des Management Service. Für Storage Server wird mit dem Paramter `i` eine ID für das Storage Target vergeben.

   ```
   /opt/beegfs/sbin/beegfs-setup-meta -p /data/beegfs/beegfs_meta \
   -s 2 -m nvm-test-01
   ```

   ```
   /opt/beegfs/sbin/beegfs-setup-storage -p /data/beegfs/beegfs_storage \
    -s 2 -i 201 -m nvm-test-01
   ```

5. Bei der Konfiguration des Clients muss nur der Management Service angegeben werden:

   ```
   /opt/beegfs/sbin/beegfs-setup-client -m nvm-test-01
   ```

6. Abschließend können die Dienste gestartet werden und das System ist einsatzbereit.

Die Installation und Konfiguration der Komponenten verlief erfolgreich und die grundlegenden Funktionen wurden erfolgreich getestet.

### Konfiguration auf den Bare-Metal Servern

In der Bare-Metal Umgebung haben die Geräte veränderte Eigenschaften, die für die Konfiguration berücksichtigt werden müssen. Im Wesentlichen unterscheidet sich die Konfiguration durch die zusätzliche Netzwerkschnittstelle und weitere Datenträger, die konfiguriert werden müssen.

Als Betriebssystem wurde AlmaLinux 8 in der Version (8.8, Kernel 4.18.0-477) verwendet. AlmaLinux ist, wie CentOS, ein Derivat von Red Hat Enterprise Linux.

Der Fernzugriff auf die Server erfolgt über die Remote Management Server, die mit dem Netzwerk des Rechenzentrums verbunden sind. Über diese können Eingaben für die Installation des Betriebssystems per Remote Console erfolgen.

Per Remote Management Software[2] oder Preboot Execution Environment (PXE) wird der Installationsdatenträger für das Betriebssystem eingehängt.

Die Installation des Betriebssystems erfolgt per Remote Console. Dabei erfolgt die Konfiguration der Netzwerkschnittstelle.

Wie in Abschnitt 2.1 beschrieben, werden zwei 1 Gb/s-Netzwerkschnittstellen als Bond aggregiert, welche über das Netzwerk im Rechenzentrum mit dem Internet verbunden sind.

Nach der Installation des Betriebssystems, kann die weitere Konfiguration per SSH erfolgen.

Nach der Installation des Betriebssystems, soll die Installation von BeeGFS mit verschiedenen Konfigurationen des Dateisystems, wie vorher in Abschnitt 2.4.4 beschrieben, erfolgen. Im nächsten Abschnitt wird beschrieben wie mit Hilfe des Tools *Ansible* die Installation automatisiert werden kann.

[2] IPMIView - https://www.supermicro.com/en/solutions/management-software/ipmi-utilities

## 3.2 Installation per Ansible Playbooks

Damit die Installation des Systems nicht mehrfach händisch ausgeführt werden muss, sollen die Installation und Konfiguration möglichst automatisiert werden.

**Ansible**[3] ist ein verbreitetes Tool, mit dem die Automatisierung erfolgen kann.

Mit sogenannten **Playbooks** können Entwürfe von Skripten erstellt werden, die automatisiert Aufgaben auf mehreren Systemen durchführen. Mit den Entwürfen können die verschiedenen Konfigurationen automatisiert auf den Servern installiert oder auch Updates ausgeführt werden.

Ansible kann von einem entfernten Host ausgeführt werden und benötigt keine zusätzliche Client Software. Der Zugriff funktioniert per SSH-Protokoll.

Wenn Server mit identischer Konfiguration installiert werden sollen, kann ein Playbook gleichzeitig auf mehreren Servern ausgeführt werden und die repetitive Arbeit während der Installation und Konfiguration abnehmen. Dadurch minimiert sich die Chance, dass bei manueller Ausführung Fehler auftreten. Das Tool gibt dem Anwender Feedback, ob einzelne Tasks (Skript-Abschnitte) erfolgreich ausgeführt wurden, oder ein unerwarteter Rückgabestatus während einzelner Befehle auftrat.

Mit einem erstellten "Bootstrap"-Skript wird auf den Servern ein neuer Benutzer für die Verwaltung mit Ansible erstellt und ein SSH-Key für den Zugriff eingerichtet.

Mithilfe von Variablen werden für jeden Server Eigenschaften für die Konfiguration vorgegeben. Im Wesentlichen werden die ID's für die Installation von den BeeGFS-Komponenten konfiguriert, IP-Adressen festgelegt und Variablen für die Wahl der Dateisystemkonfiguration angegeben.

In der *Inventory*-Datei wird in einer Liste angegeben, welche BeeGFS-Komponenten auf den Servern installiert werden sollen (Gruppierung in *Management*-, *Metadata*-, *Storage*-Komponente oder *Client*).

Das erstellte Skript "install_beegfs" startet die eigentliche Konfiguration und Installation.

[3] Ansible Automatisierungssoftware - https://www.ansible.com/

**Ablauf:**

1. Es werden auf allen Servern die BeeGFS-Repositories heruntergeladen, das Shared Secret für die Kommunikation übertragen, notwendige Pakete für das Bauen von Kernel-Modulen installiert und bestehende Software-RAID Dateisysteme auf den Datenträgern gelöscht.
2. Entsprechend einer Variable wird ein Skript für die Datenträgerkonfiguration ausgeführt und das Dateisystem eingehängt.
3. Für die zwei 100 Gb/s Schnittstellen wird ein Bond erstellt und die Schnittstelle wird konfiguriert.
4. Entsprechend der *Inventory*-Datei wird auf den Hosts, die Installation und Konfiguration der Komponenten ausgeführt:
    - Metadata Server Installation
    - Storage Server Installation
    - Management Server Installation
    - Client Installation
5. Abschließend werden alle Dienste gestartet.

Das System ist nun einsatzbereit und die installierten Clients können auf das Dateisystem zugreifen.

Für die Evaluierung ist zusätzlich die Installation der Anwendungen MPI und IOR erforderlich. Da die Installation optional ist, folgt mit einem weiteren Ansible Playbook die Installation der Benchmark-Tools.

Anschließend können von einem Server ausgehend parallel auf allen Servern Benchmarks ausgeführt werden.

Nachdem die Durchführung der Benchmarks für eine Konfiguration abgeschlossen ist, wird das System mit einer anderen Konfiguration neu installiert.

### Dokumentation

In einer Readme wird für die Mitarbeiter der Projektgruppe im Detail beschrieben, wie die Einrichtung der Server und Ausführung der Skripte erfolgt (siehe Anhang ab A.3).

## 3.3 Benchmarks

Wie in Abschnitt 2.5 erwähnt, werden verschiedene Benchmarks für die unterschiedlichen Kompetenzen (Durchsatz-Geschwindigkeit und Metadaten-Leistung) ausgeführt.

Um die Benchmarks auszuführen, wurden, wie in Abschnitt 3.2 beschrieben, die notwendigen Anwendungen installiert und die Server als Clients installiert. Über eine Konfigurationsdatei kann angegeben werden, auf welchen Servern die Benchmarks ausgeführt werden sollen.

Im Folgenden wird beschrieben, mit welchen Befehlen und Argumenten die Ausführung der Benchmarks erfolgt.

### 3.3.1 mpirun

Wie in Abschnitt 2.5 beschrieben, basieren die Anwendungen auf dem Standard *MPI*. Folgender Befehl ist Grundlage für das Ausführen der parallelen Benchmarks.

```
mpirun -hostfile /tmp/nodefile --map-by node -np ${no_of_clients} ...
```

In der Datei `nodefile` sind die Server angegeben, auf denen der Test ausgeführt werden soll. Das Tool verwendet für die Kommunikation SSH. Mit `${no_of_clients}` wird angegeben, wie viele Threads parallel gestartet werden.

### 3.3.2 Multi-stream throughput - Benchmark

Im *Multi-stream throughput* Benchmark wird eine Datengröße gewählt, die für optimale Durchsatz-Geschwindigkeit ausgelegt ist, indem sie genau der oder einem Vielfachen der Blockgröße (Chunksize) von dem BeeGFS System entspricht.

Befehl:

```
mpirun -hostfile /tmp/nodefile --map-by node -np ${no_of_clients}\
ior -wr -i5 -t2m -b ${block_size} -g -F -e -o /mnt/beegfs/test.ior
```

Dabei entspricht `no_of_clients` der Anzahl der Threads (von 1 bis 256). Die Option `b` mit dem Argument `block_size` entspricht der Datengröße pro Prozess. Die `block_size` wird mit `((3 * RAM_PER_SERVER * NUM_SERVERS) / ${NUM_PROCS})` errechnet. Dabei entspricht `RAM_PER_SERVER` der Größe des Arbeitsspeichers pro Server und `NUM_SERVERS` der Anzahl der Server.

- `-wr` Schreib- und Lesetest
- `-i` Anzahl der Iterationen
- `-t` Größe pro Datenblock
- `-b` Daten-Übertragungsmenge des Benchmarklaufs (vielfaches von `-t`)
- `-g` use barriers between open, write/read, and close
- `-F` file-per-process
- `-e` perform fsync upon POSIX write close
- `-o` full name for test file (Pfad)

(Vgl. [32])

Der Benchmark wird für eine Datenblockgröße 2 MiB bzw. 2048 KiB und Datenmenge von 1,50 TiB mit fünf Iterationen durchgeführt.

Betrachtet werden im Ergebnis die durchschnittlichen Werte der Übertragungsgeschwindigkeit in MiB pro Sekunde für das Schreiben und Lesen.

### 3.3.3 Shared file throughput - Benchmark

Im *Shared file throughput* Benchmark wird eine Datengröße gewählt, die nicht der Chunksize des Systems entspricht. Damit wird ein realistischeres Szenario abgebildet, wie in einem Deep Learning Dataset, bei dem Dateigrößen ungleich der Chunksize der Regelfall sind.

Befehl:

```
mpirun -hostfile /tmp/nodefile --map-by node -np ${no_of_clients}\
ior -wr -i5 -t1200k -b ${block_size} -g -e -o /mnt/beegfs/test.ior
```

Der Benchmark wird für eine Datenblockgröße 1200 KB und Datenmenge von 1,17 TiB mit fünf Iterationen durchgeführt.

Die Argumente und Optionen sind analog zu Abschnitt 3.3.2.

### 3.3.4 Metadaten - Benchmark

Folgender Befehl wird für die Ausführung der Metadaten-Benchmarks verwendet:

```
mpirun -hostfile /tmp/nodefile --map-by node -np ${no_of_clients}\
mdtest -d /mnt/beegfs/mdtest -i 2 -I ${no_of_files}\
-z 1 -b ${no_of_dirs} -L -u -F
```

Dabei entspricht `no_of_clients` der Anzahl der Threads (von 1 bis 256). Die Option `I` mit dem Argument `no_of_files` entspricht der Anzahl zu erstellender Dateien pro Prozess und ist in diesem Test fest auf 1024 gesetzt. Entsprechend wird der Test für 1048576 Operationen ausgeführt. Die Option `b` mit dem Argument `no_of_dirs` beschreibt, wie viele Ordner jeder Prozess erstellt. Die Anzahl ergibt sich aus 1024/`no_of_clients`.

- `-i` Anzahl der Iterationen
- `-I` Anzahl der Dateien pro Verzeichnis
- `-b` Anzahl der Ordner pro Unterverzeichnis
- `-z` Ordnertiefe (`-b` wird pro Ordnertiefe ausgeführt)

Betrachtet werden im Ergebnis die durchschnittlichen Werte der Operationen pro Sekunde für `File creation`, `File stat` und `File read`.

- `File creation` - Beschreibt, wie viele Dateien pro Sekunde erstellt werden.
- `File stat` - Beschreibt, wie viele Datei-Status Operationen pro Sekunde zurückgegeben werden (letzter Zugriff, letzte Modifikation, letzter Zeitpunkt der Änderung...).
- `File read` - Beschreibt, wie viele Lese-Operationen pro Sekunde ausgeführt werden.

### 3.3.5 IOPS - Benchmark

Bei dem *IOPS*-Benchmark wird eine Datengröße gewählt, die sehr klein (4 KiB) ist. Das entspricht einem Metadaten-Test bei dem im Vergleich zu `mdtest` Daten mit mehr als 0 b erstellt werden.

Befehl:

```
mpirun -hostfile /tmp/nodefile --map-by node -np ${no_of_clients}\
ior -w -i5 -t4k -b ${block_size} -F -z -g -o /mnt/beegfs/test.ior
```

Der Benchmark wird für eine Datenmenge von 150 GiB (75 GiB für die Testläufe mit ein und zwei Threads) mit fünf Iterationen durchgeführt.

Die Argumente und Optionen sind analog zu den Benchmarks in 3.3.2.

Die in diesem Kapitel aufgeführten Varianten der Benchmarks dienen als Grundlage für die Evaluation, die in Kapitel 4 folgt.

# 4 Evaluation

In diesem Kapitel erfolgt die Evaluation entsprechend der in Abschnitt 3.3 beschriebenen Benchmarks.

Mit Hilfe der Automatisierungsskripte für die Installation, wie in Abschnitt 3.2 beschrieben, wird die Installation der verschiedenen Konfigurationen und das Ausführen der Benchmarks auf den Bare-Metal Servern ausgeführt. Für jede Installation werden die beschriebenen Benchmarks ausgeführt und in der Evaluation soll ermittelt werden, welche Konfiguration am besten für den Einsatz als Zwischenspeicher für Deep Learning Anwendungen geeignet ist.

Im Abschnitt 4.1 werden die verschiedenen Testkonfigurationen und deren Eigenschaften aufgeführt, die in dieser Arbeit untersucht werden. Die Durchsatz-Ergebnisse werden im Abschnitt 4.2 für die Szenarien *Multi-stream throughput* und *Shared file throughput* analysiert. Dabei wird festgehalten, welche der Konfigurationen die beste Durchsatz-Leistung erzielt. Im Abschnitt 4.3 wird die Fähigkeit der Skalierung hinsichtlich der Storage-Leistung beurteilt und der Einfluss der gleichzeitigen Ausführung der Benchmarks auf das System bewertet. Am Ende des Kapitels wird die Metadaten-Leistung der unterschiedlichen Konfigurationen verglichen. Dabei soll untersucht werden, inwiefern sich die Leistung mit separaten Laufwerken für Metadata Server von Konfigurationen mit geteilten Laufwerken unterscheidet. Abschnitt 4.4 zeigt die Ergebnisse der Metadaten-Benchmarks.

## 4.1 Testkonfigurationen

Es wurden acht verschiedene Arten von Konfigurationen getestet, in denen entweder ein separates Laufwerk für Metadaten genutzt wird oder ein Software-RAID als gemeinsames Laufwerk für Storage und Metadaten.

Grundsätzlich sind die Server symmetrisch konfiguriert. Das heißt, auf allen Nodes wurde die gleiche Konfiguration für Storage und Metadaten gewählt. Eine der Konfigurationen unterscheidet sich zum Vergleich, indem nur einer der Nodes als Metadata Server konfiguriert ist.

Das Betriebssystem der Testkonfigurationen ist AlmaLinux 8.8 (Kernel 4.18.0-477) und BeeGFS wurde in der Version 7.4.0 verwendet.

Als grundlegendes Dateisystem für Metadaten-Laufwerke wurde entsprechend der Dokumentation (vgl. [48]) ext4 verwendet. Die Dokumentation beruht auf den Erkenntnissen zur Evaluation verschiedener Dateisysteme als Metadaten-Dateisystem [21].

Für den Storage wurde nach der Empfehlung der Dokumentation (vgl. [52]) XFS als Dateisystem und mdadm oder ZFS als Software-RAID gewählt. Dabei ist mdadm eine Software die verwendet wird um ein Verbund aus Laufwerken als Block Device zu konfigurieren, wohingegen ZFS ein eigenes Dateisystem mit zusätzlichen Funktionalitäten wie Software-RAID ist.

Die Größe der Metadatenlaufwerke, ist anhand der Anzahl der Dateien zu wählen und unterscheidet sich je nach grundlegendem Dateisystem. Laut Dokumentation [53] sollten mindestens 0,3% der Dateisystemkapazität für Metadaten verfügbar sein.

Die kombinierte maximale Datenträgerkapazität beträgt etwa 140 TiB (20 Laufwerke je 7,68 TB). Ein Teil der Kapazität steht je nach Konfiguration durch Paritätslaufwerke oder separate Metadatenlaufwerke nicht als Storagekapazität zur Verfügung.

Es folgt die Erläuterung der Besonderheiten der Konfigurationen und anschließend in Tabelle 4.1 eine detaillierte Auflistung aller Konfigurationen und Eigenschaften.

Konfigurationen mit ZFS wurden, falls nicht ein separates Laufwerk für Metadaten verwendet wird, mit ZFS-Datasets (ZFS-Dateisysteme) in je ein Unterverzeichnis für Storage und Metadaten aufgeteilt und mit Quotas begrenzt.

Bei der Konfiguration *ZFS-s-1-mds* wird nur auf einem Node der Metadata Service ausgeführt. Das Metadaten-Laufwerk wurde mit ext4 als Dateisystem formatiert.

Die Konfiguration *multi-target* verwendet kein Software-RAID. Es werden vier Laufwerke mit XFS formatiert und mit eigenen System-Einhängepunkten eingebunden. Der Storage Service wird anstatt mit einem Storage Target mit vier Zielverzeichnissen konfiguriert.

Es übernimmt der Storage Service jedes Nodes die Aufgabe die Daten über die Storage Targets aufzuteilen (Striping).

Bei der Konfiguration *multi-mode* werden, wie in der vorherigen Konfiguration, vier Laufwerke mit XFS formatiert und eingehängt. Es wird pro Node und XFS-Laufwerk ein Storage Service erstellt. Jeder Storage Service erhält eine eigene Konfiguration mit separatem Storage Target. Es übernimmt nun der Metadata Service die Aufgabe, die Dateien über in diesem Fall 16 Storage Services zu stripen.

Für die Konfiguration *mdadm* wurden in der Testkonfiguration keine Quotas konfiguriert, was mit XFS als Dateisystem jedoch auch möglich ist.

Im Folgenden werden im Text meist die Konfigurationen *ZFS-s, ZFS-z, ZFS-s-ext4, ZFS-z-ext4* und *ZFS-s-1-mds* als "Konfigurationen mit ZFS" oder *multi-target, multi-mode* und *mdadm* als "Konfigurationen mit XFS" zusammengefasst, falls nicht anders angegeben.

In den Diagrammen werden für die verschiedenen Konfigurationen die Abkürzungen entsprechend der Tabelle 4.1 verwendet.

Für den Vergleich zum bestehenden NFS-Server, wurden die Durchsatz-Benchmarks und Metadaten-Benchmarks ebenso auf einer virtuellen Maschine im Rechenzentrum des FTZ SMSY ausgeführt. In den Diagrammen werden die Werte mit der Bezeichnung *NFS-Server* abgebildet.

| Abk. | Konfiguration | RAID | Kapazität | Metadaten | Bemerkung |
|---|---|---|---|---|---|
| ZFS-s | ZFS Striped | RAID0 | 132,8 TiB | ZFS-Dataset mit etwa 2% Quota | kein Paritätslaufwerk |
| ZFS-z | ZFS RAID-Z | RAID5 | 103,5 TiB [1] | ZFS-Dataset mit etwa 3% Quota | ein Paritätslaufwerk |
| ZFS-s-ext4 | ZFS Striped, Metadaten separat | RAID0 | 103,5 TiB [2] | ext4; auf je einem Laufwerk | kein Paritätslaufwerk, etwa 20% Kapazität für Metadaten |
| ZFS-z-ext4 | ZFS RAID-Z, Metadaten separat | RAID5 | 80 TiB [1], [2] | ext4; auf je einem Laufwerk | ein Paritätslaufwerk, etwa 33,3% Kapazität für Metadaten |
| ZFS-s-1-mds | ZFS Striped, **ein** Metadatenserver mit separatem Laufwerk | RAID0 | 125,5 TiB [3] | ext4; auf einem Laufwerk | kein Paritätslaufwerk, etwa 5% Kapazität für Metadaten |
| multi-target | XFS, Multiple Targets, Metadaten separat | - | 111,7 TiB [2] | ext4; auf je einem Laufwerk | XFS-Dateisystem für Storage, etwa 25% Kapazität für Metadaten, kein Paritätslaufwerk |
| multi-mode | XFS, Multi Mode, Metadaten separat | - | 111,7 TiB [2] | ext4; auf je einem Laufwerk | XFS-Dateisystem für Storage, etwa 25% Kapazität für Metadaten, kein Paritätslaufwerk |
| mdadm | mdadm mit XFS als Dateisystem | RAID0 | 139,7 TiB | kein Quota konfiguriert | XFS-Dateisystem für Storage und Metadaten, kein Paritätslaufwerk |

Tabelle 4.1: Testkonfigurationen im Vergleich

[1] Storage-Kapazität um 1/5 reduziert (ein Paritäts-Laufwerk pro Node).
[2] Storage-Kapazität um 1/5 reduziert (ein separates Metadaten-Laufwerk pro Node).
[3] Storage-Kapazität um 1/20 reduziert (ein separates Metadaten-Laufwerk).

## 4.2 Durchsatz-Benchmarks

Im Folgenden werden die Diagramme der Anwendungsfälle *Multi-stream throughput* und *Shared file throughput* für Lese- und Schreibvorgänge aufgeführt und bewertet.

### Multi-stream throughput

In diesem Benchmark werden die Durchsatz-Geschwindigkeiten mit 2 MiB Datenblöcken und insgesamt 1,50 TiB Datenmenge getestet (vgl. Abschnitt 3.3.2).

In Abbildung 4.1 kann beobachtet werden, dass die Konfigurationen mit XFS beim Lesen mit weniger als 32 Threads bis zu 35% langsamer und ab höherer Anzahl an Threads tendenziell gleichauf mit ZFS sind. In der Konfiguration *multi-mode* sind die schnellsten Lesegeschwindigkeiten mit 128 und 256 Threads mit bis zu 25% höherer Durchsatz-Leistung gegenüber anderen Konfigurationen festzustellen.

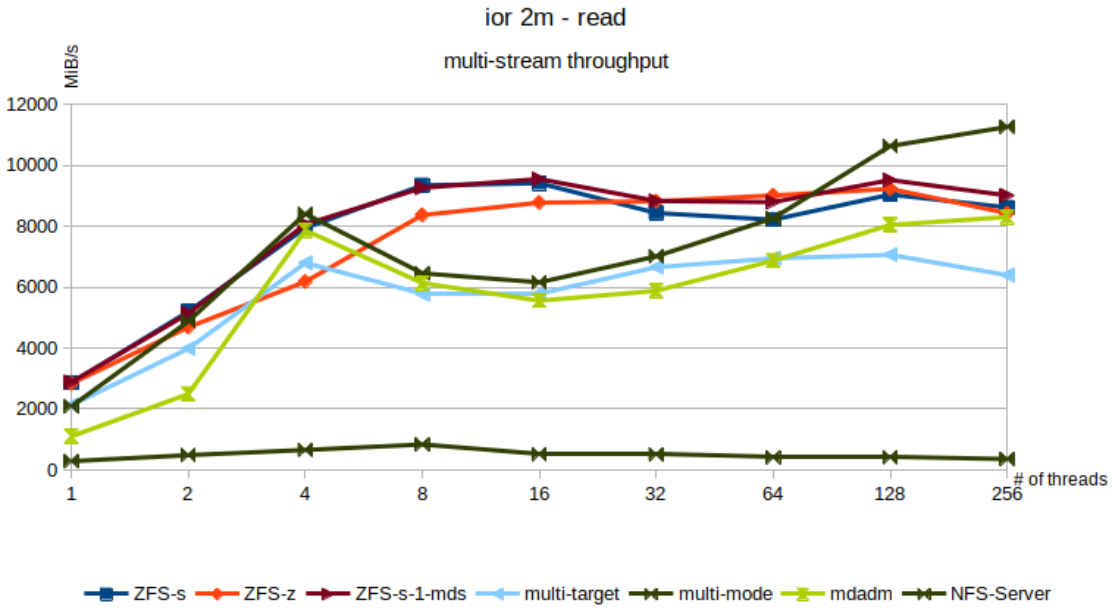


Abbildung 4.1: *Multi-stream throughput* (2 MiB) – Lesevorgang
(Abb. ohne *ZFS-s-ext4* und *ZFS-z-ext4* - vergleichbar mit *ZFS-s* und *ZFS-z*)

Abbildung 4.2 zeigt beim Schreiben, dass die Konfigurationen mit XFS mit weniger als 32 Threads bis zu 25% schneller als die Konfigurationen mit ZFS sind. Ab einer höheren Anzahl an Threads tendenziell gleichauf.

In der Konfiguration *multi-mode* sind die schnellsten Schreibgeschwindigkeiten (128 und 256 Threads) mit bis zu 25% Vorsprung und maximal 10,8 GiB/s festzustellen.

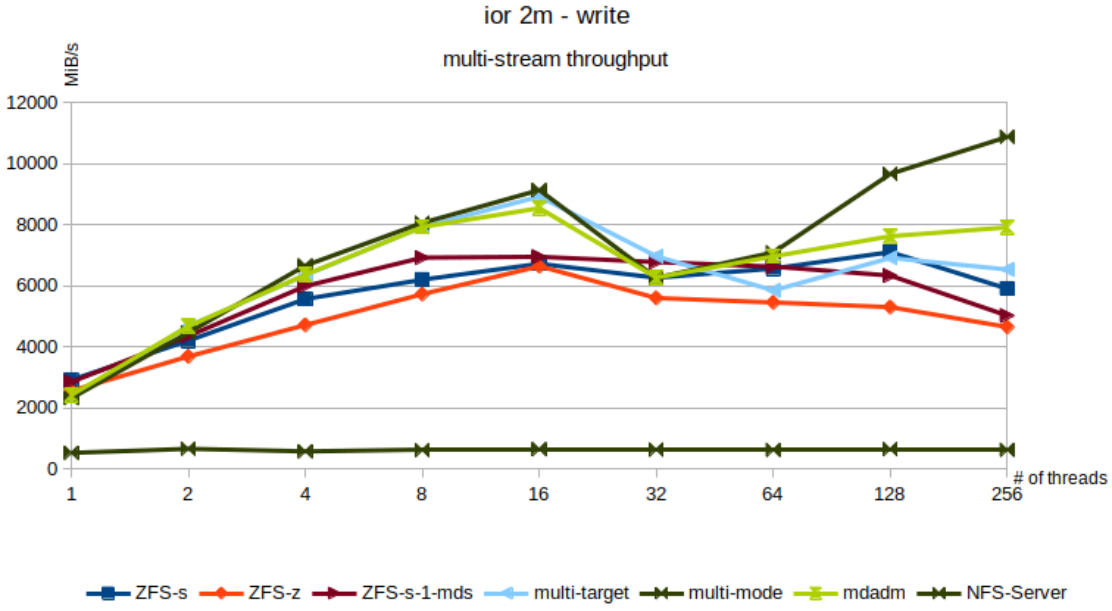


Abbildung 4.2: *Multi-stream throughput* (2 MiB) – Schreibvorgang
(Abb. ohne *ZFS-s-ext4* und *ZFS-z-ext4* - vergleichbar mit *ZFS-s* und *ZFS-z*)

Die Konfigurationen mit ZFS Striped zeigen durchschnittlich 10% höhere Durchsatz-Leistung als Konfigurationen mit ZFS RAID-Z.

Mit dem bestehenden NFS-Server wurden beim Lesen maximal 836 MiB/s und beim Schreiben maximal 655 MiB/s gemessen.

### Shared file throughput

Im *Shared file throughput*-Benchmark werden die Durchsatz-Geschwindigkeiten mit 1,2 MB Datenblöcken und insgesamt 1,17 TiB Datenmenge getestet (vgl. Abschnitt 3.3.3).

Es kann beobachtet werden, dass die Konfigurationen mit ZFS untereinander ähnlich schnell sind. ZFS mit RAID-Z ist, wie im Benchmark *Multi-stream throughput*, durchschnittlich 10% langsamer.

Die Konfigurationen mit XFS sind beim Lesen und Schreiben in diesem Testfall durchschnittlich 30% langsamer.

Mit dem bestehenden NFS-Server ist der maximale Durchsatz beim Lesen 446 MiB/s und beim Schreiben 672 MiB/s.

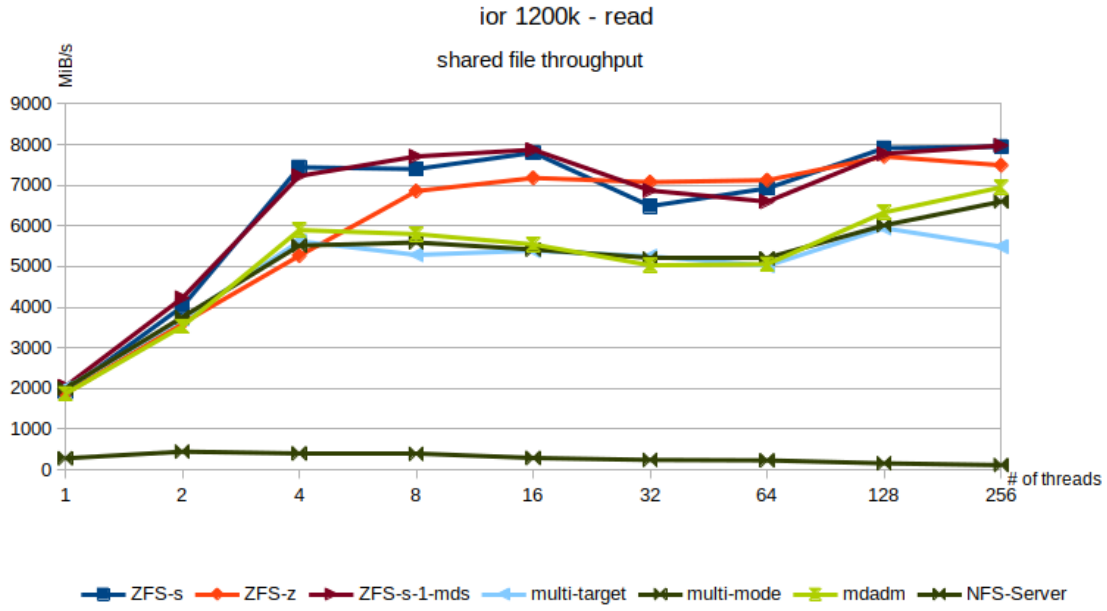


Abbildung 4.3: *Shared file throughput* (1,2 MB) – Lesevorgang
(Abb. ohne *ZFS-s-ext4* und *ZFS-z-ext4* - vergleichbar mit *ZFS-s* und *ZFS-z*)

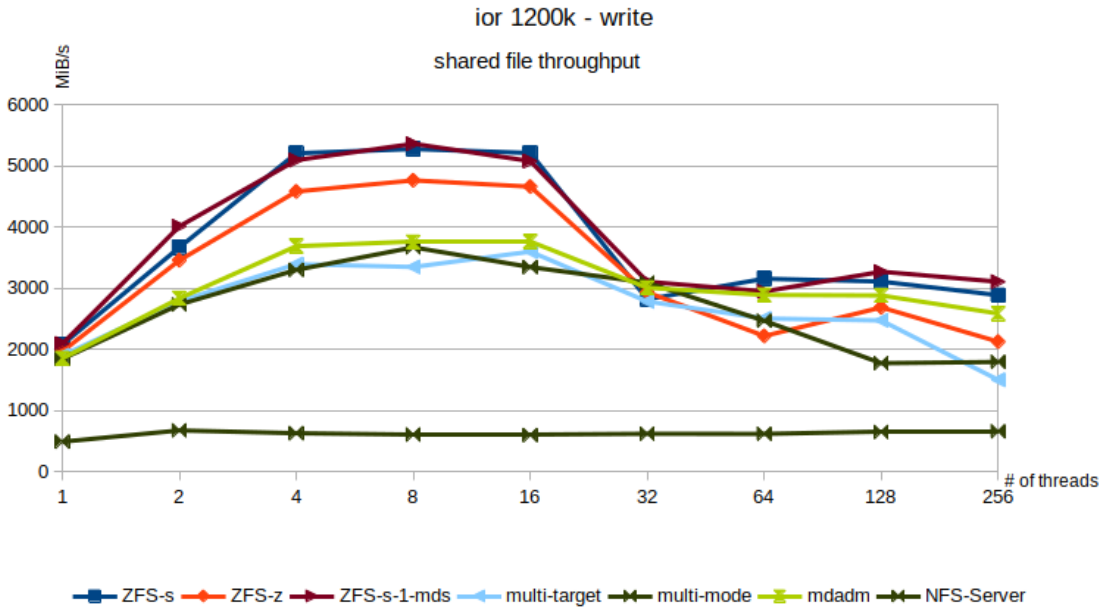


Abbildung 4.4: *Shared file throughput* (1,2 MB) – Schreibvorgang
(Abb. ohne *ZFS-s-ext4* und *ZFS-z-ext4* - vergleichbar mit *ZFS-s* und *ZFS-z*)

### Zwischenfazit der Durchsatz-Benchmarks

Es kann festgehalten werden, dass im *Multi-stream throughput* Konfigurationen mit XFS den höchsten Durchsatz mit etwa 11 GiB/s erreichen. Beim Lesen sind Konfigurationen mit ZFS vor allem bei niedriger Anzahl an Threads durchschnittlich 22% schneller, während beim Schreiben die Konfigurationen mit XFS durchschnittlich 26% höhere Durchsatz-Geschwindigkeiten erreichen. Dieser Benchmark entspricht einem synthetischen Benchmark und ist nicht repräsentativ für einen realen Anwendungsfall.

Im Szenario *Shared file throughput* zeigt sich, dass Konfigurationen mit ZFS beim Lesen Durchsatz-Geschwindigkeiten von ca. 8 GiB/s und beim Schreiben Durchsatz-Geschwindigkeiten von 5,2 GiB/s erreichen. Ein Leistungsvorsprung von durchschnittlich 21% zu Konfigurationen mit XFS kann festgestellt werden.

Gegenüber der Leistung des bestehenden NFS-Servers ist der Durchsatz im *Shared file throughput* beim Lesen um den Faktor 17 und beim Schreiben um den Faktor 7 höher.

Es ist festzustellen, dass die Ergebnisse mit den höchsten Werten oft bei einem Testfall mit 16 Threads auftreten. Bei vielen Konfigurationen ist zu erkennen, dass die Leistung mit zunehmender Anzahl von Threads abfällt.

Ein Grund für sich annähernde Ergebnisse bei höherer Anzahl an Threads kann eine höhere CPU-Belastung durch ZFS sein, wie sich auch in der CPU-Auslastung (vgl. Abbildung A.5) widerspiegelt.

Grundsätzlich kann die abfallende Leistung bei höherer Anzahl an Threads durch CPU-Belastung, die durch das gleichzeitige Ausführen der Benchmarks verursacht wird, vermutet werden.

Im Folgenden Abschnitt soll analysiert werden, wie gut das System mit zunehmender Anzahl an Nodes skaliert, und ob bei höherer Anzahl an Threads Leistungseinbußen durch Ausführen der Benchmarks festzustellen sind.

## 4.3 Skalierungs-Test

In diesem Abschnitt werden Ergebnisse der Benchmarks von der Konfiguration *ZFS-s* mit unterschiedlicher Anzahl von Nodes verglichen, um festzustellen, wie gut die Lese- und Schreibleistung des Systems mit steigender Anzahl von Nodes skaliert. In Abschnitt 4.3.2 werden im Nachhinein potenzielle Leistungseinbußen durch gleichzeitiges Ausführen der Server-Komponenten und der Benchmarks (Client) untersucht.

### 4.3.1 Skalierung des Systems

Es werden Lese- und Schreibgeschwindigkeiten von einer Konfiguration mit einem Node, zwei Nodes, drei Nodes und vier Nodes verglichen. Dabei werden jeweils unbenutzte Nodes nur als Clients konfiguriert, um den Benchmark auszuführen.

### read-Skalierung

Die Ergebnisse zeigen, dass beim Lesevorgang im Fall mit den höchsten Durchsatzgeschwindigkeiten (16 Threads) das System mit 2 Nodes mit 89%, bei 3 Nodes mit 66% und bei 4 Nodes mit 48% skaliert. Es ist zu beachten, dass die Systemlast durch Ausführung des Benchmarks auf den Nodes entsprechend höher wird.

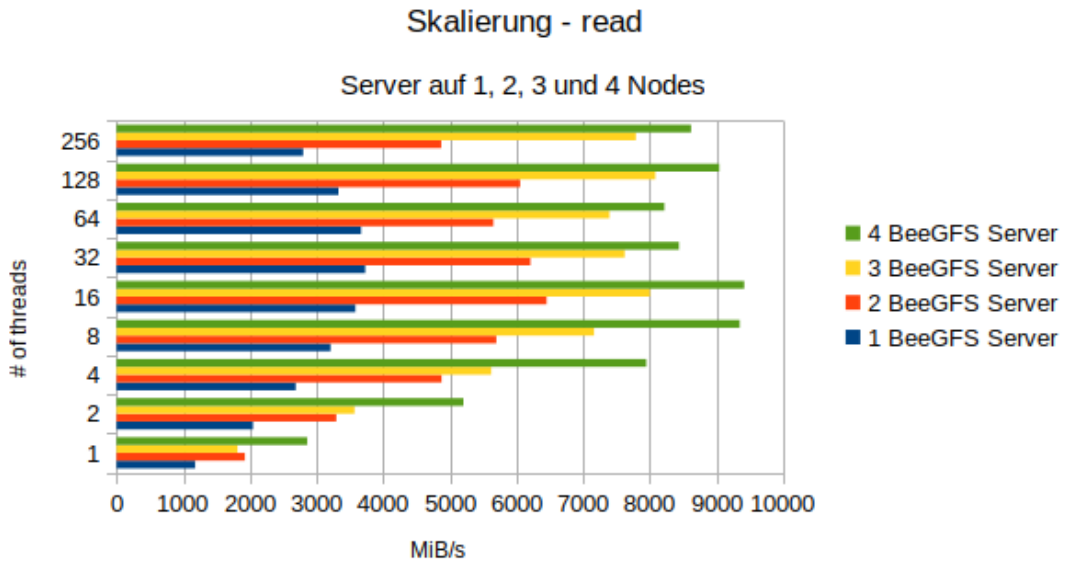


Abbildung 4.5: Skalierung ZFS Striped (2 MiB) – Lesevorgang

### write-Skalierung

Die Ergebnisse zeigen, dass beim Schreibvorgang im Fall mit 16 Threads das System mit 2 Nodes mit 79%, bei 3 Nodes mit 47% und bei 4 Nodes mit 25% skaliert. Auch mit 128 Threads skaliert das System vergleichsweise gut mit 2 Nodes mit 90%, bei 3 Nodes mit 83% und bei 4 Nodes mit 80%.

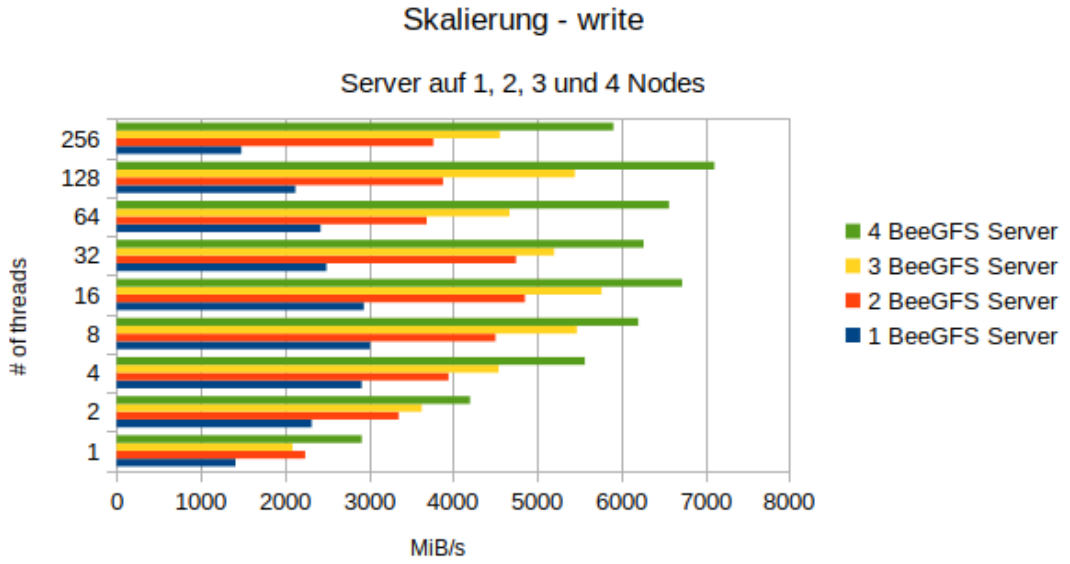


Abbildung 4.6: Skalierung ZFS Striped (2 MiB) – Schreibvorgang

### 4.3.2 Systemlast durch Benchmarks

In den vorherigen Ergebnissen ist der Einfluss der Systemlast durch das Ausführen der Benchmarks auf den Nodes nicht berücksichtigt. Die Ergebnisse sind daher nur beschränkt und untereinander vergleichbar.

Da die Anzahl der verfügbaren Testsysteme durch die Gegebenheiten auf die vier Server beschränkt ist, wurden die Benchmarks bisher gleichzeitig auf den Servern ausgeführt.

Im Folgenden Test wird der Server auf einem, zwei oder drei Nodes ausgeführt, während die übrigen Nodes als Clients konfiguriert sind und nur die Benchmarks ausführen.

Es werden die Lese- und Schreibgeschwindigkeiten als Beispiel bei einer Konfiguration mit zwei Nodes als Server in den Diagrammen 4.7 und 4.8 abgebildet. Die Ergebnisse berücksichtigen zusammenfassend auch die Werte von den Konfigurationen mit einem und drei Nodes.

In der Konfiguration sind nur die zwei nicht als Server verwendeten Nodes als Clients für Benchmarks konfiguriert. Zum Vergleich werden die Werte aus dem vorherigen Abschnitt abgebildet (mit gleichzeitiger Ausführung der Benchmarks auf den Server-Nodes).

Die Tests wurden auch mit einem und drei Nodes als Server wiederholt. Im Anhang sind die Diagramme der weiteren Testfälle zu finden (ein Node - Lesevorgang A.1, drei Nodes - Lesevorgang A.2, ein Node - Schreibvorgang A.3 und drei Nodes - Schreibvorgang A.4).

Wie in Abschnitt 4.2 festgestellt wurde, sinkt die Durchsatz-Leistung bei Benchmarks mit mehr als 16 Threads. Die folgenden Ergebnisse zeigen, dass bei Testfällen mit höherer Anzahl an Threads, erhebliche Leistungsdifferenzen festzustellen sind.

Die Diagramme zeigen die Ergebnisse, wenn die Server-Nodes nicht die Benchmarks ausführen (*2 Server als Client (exkl. BeeGFS-Server)*) und wenn die Server-Nodes gleichzeitig als Clients Benchmarks ausführen (*4 Server als Client (inkl. BeeGFS-Server)*).

### read-Benchmark

Abbildung 4.7 zeigt, dass beim Lesen in Testfällen mit höherer Anzahl an Threads Leistungsunterschiede von bis zu 31% festgestellt werden können. Die Lesegeschwindigkeiten können durchschnittlich 12% höher eingeschätzt werden.

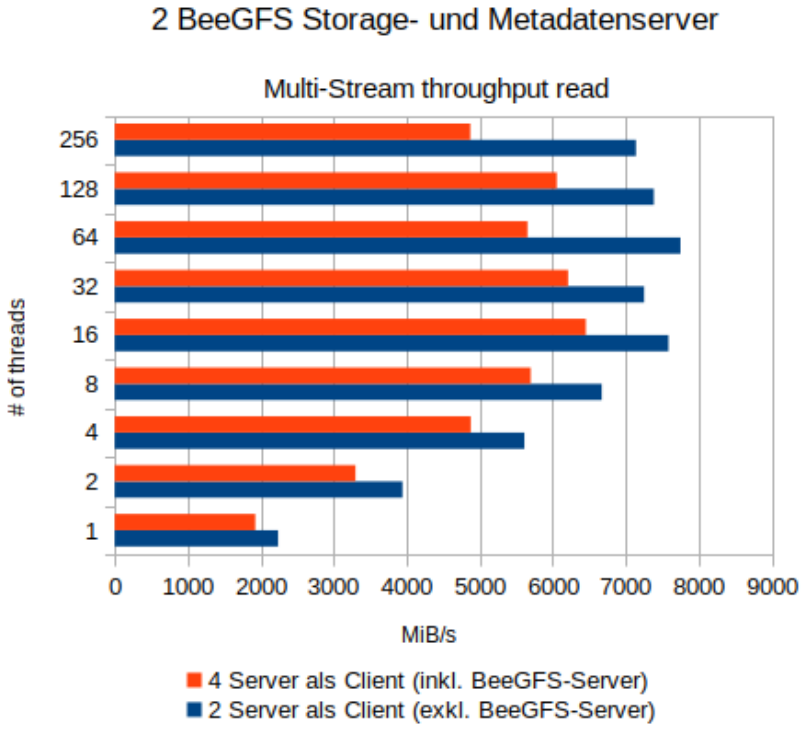


Abbildung 4.7: Systemlast 2 Server ZFS Striped (2 MiB) – Lesevorgang

### write-Benchmark

Abbildung 4.8 zeigt, dass beim Schreiben durchschnittlich Leistungsunterschiede von 22% festgestellt werden können.

In einzelnen Testfällen können Leistungsunterschiede bis zu 56% festgestellt werden.

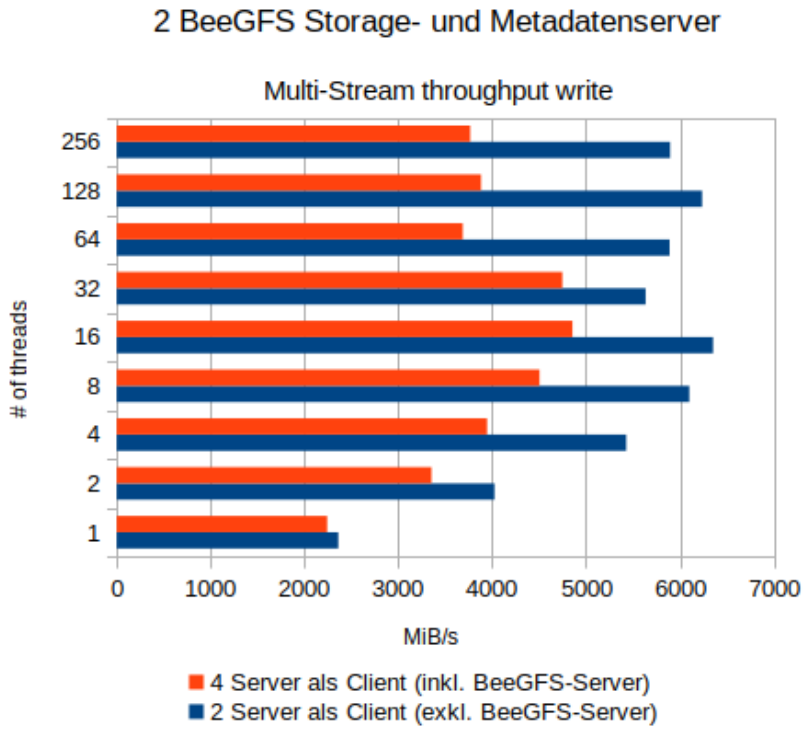


Abbildung 4.8: Systemlast 2 Server ZFS Striped (2 MiB) – Schreibvorgang

### Zwischenfazit – Skalierung und Systemlast

Im Vergleich können bessere Durchsatz-Geschwindigkeiten festgestellt werden, wenn der Benchmark nicht simultan auf dem Server ausgeführt wird.

Die Leistungseinbußen können durch diesen Vergleich auf die System-Auslastung durch die Benchmarks zurückgeführt werden.

Besonders beim Schreiben ist die Leistung durch die Ausführung der Benchmarks stark beeinflusst.

Die Diagramme zeigen, dass, im Gegensatz zu den Ergebnissen aus Abschnitt 4.2, die Leistung mit steigender Anzahl von Threads nicht wesentlich sinkt.

Damit ist für die Konfiguration *ZFS-s* die tatsächliche Leistung vermutlich höher und die Ergebnisse im vorherigen Abschnitt 4.3.1 sind nicht repräsentativ für ein Urteil über die Skalierungs-Eigenschaften des Systems.

### read-Benchmark - korrigierte Skalierung

Im Lese-Szenario kann beobachtet werden, dass das System ohne parallel laufende Benchmarks bei 2 Nodes mit 95% skaliert, gegenüber 89%, wenn gleichzeitig die Benchmarks ausgeführt werden (16 Threads).

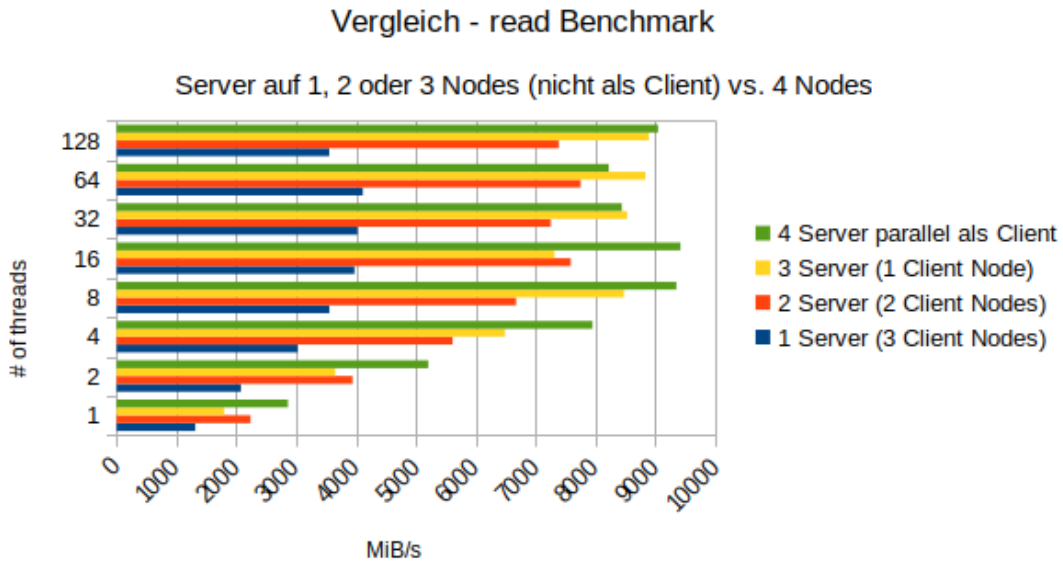


Abbildung 4.9: Skalierung “korrigiert” ZFS Striped (2 MiB) – Lesevorgang

### write-Benchmark - korrigierte Skalierung

Im Schreib-Szenario kann beobachtet werden, dass das System ohne parallel laufende Benchmarks bei 2 Nodes mit 99% skaliert, gegenüber 79%, wenn gleichzeitig die Benchmarks ausgeführt werden (16 Threads).

Verhältnismäßig schlechtere Skalierung zeigen die Benchmarks mit drei Servern und einem Node als Client.

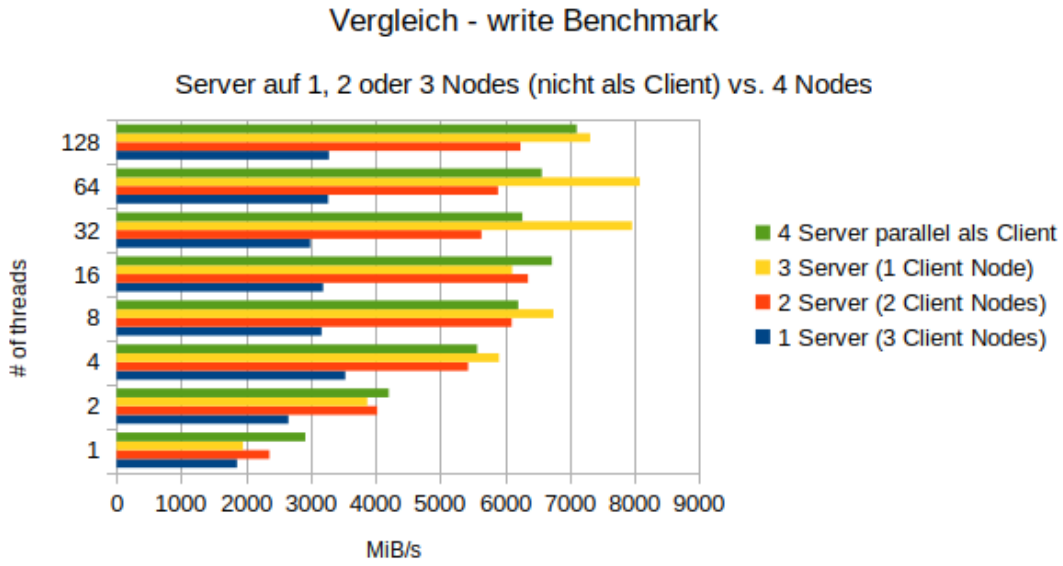


Abbildung 4.10: Skalierung "korrigiert" ZFS Striped (2 MiB) – Schreibvorgang

### Zwischenfazit – korrigierte Skalierung

Die Ergebnisse der Tests zeigen in ausgewählten Fällen verbesserte Skalierungs-Eigenschaften. Dabei ist jedoch zu beachten, dass durch die kleine Menge der Server und gleichzeitig ändernde Anzahl der Client-Nodes nur eine beschränkte Genauigkeit der Ergebnisse zu erwarten ist.

Vor allem beim Schreiben zeigen sich erhebliche Unterschiede mit etwa 20% höheren Durchsatz-Geschwindigkeiten.

## 4.4 Metadaten-Benchmark

Im Folgendem Abschnitt wird die Metadaten-Leistung untersucht. Es werden die Ergebnisse der Benchmarks mit *mdtest* und des Schreibtests mit 4 KiB-Blöcken (*IOR*-Benchmark) verglichen. Wie in Abschnitt 2.5 beschrieben, zeigen die Benchmarks, wie viele Operationen pro Sekunde auf dem Dateisystem durchgeführt werden können. Diese Eigenschaften sind ausschlaggebend für die Latenz beim Zugriff auf Dateien.

### mdtest-Benchmark

Wie in Abschnitt 3.3.4 beschrieben, werden die Werte für *File creations* pro Sekunde, *File stats* pro Sekunde und *File reads* pro Sekunde in den folgenden Diagrammen aufgeführt und die Leistung der unterschiedlichen Konfigurationen verglichen.

Die Konfiguration mit XFS und mit separatem Metadaten-Laufwerk (auch *ZFS-s-ext4* und *ZFS-z-ext4*) zeigen in den Fällen *File creations* und *File reads* pro Sekunde Ergebnisse mit bis zu 40% besserer Leistung.

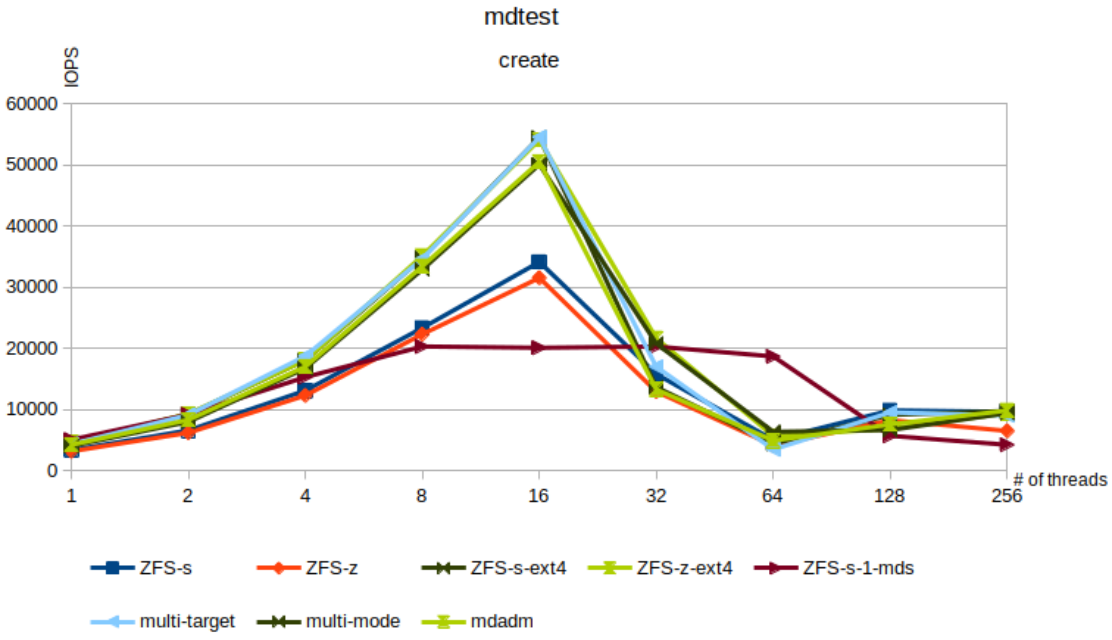


Abbildung 4.11: Metadaten – *File creations* pro Sekunde

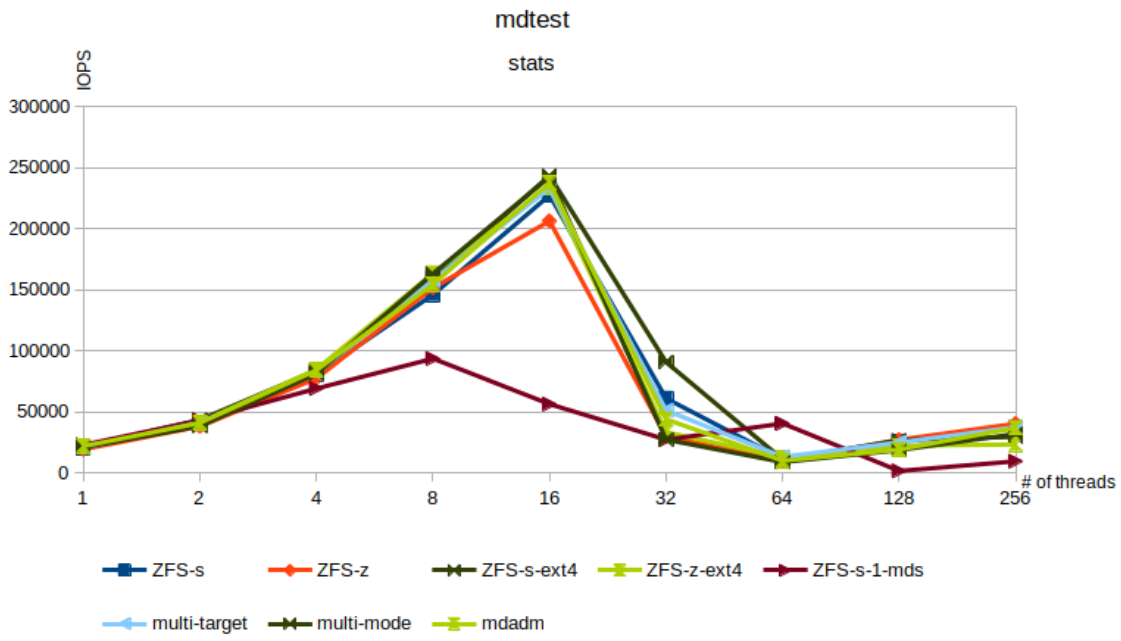


Abbildung 4.12: Metadaten – *File stats* pro Sekunde

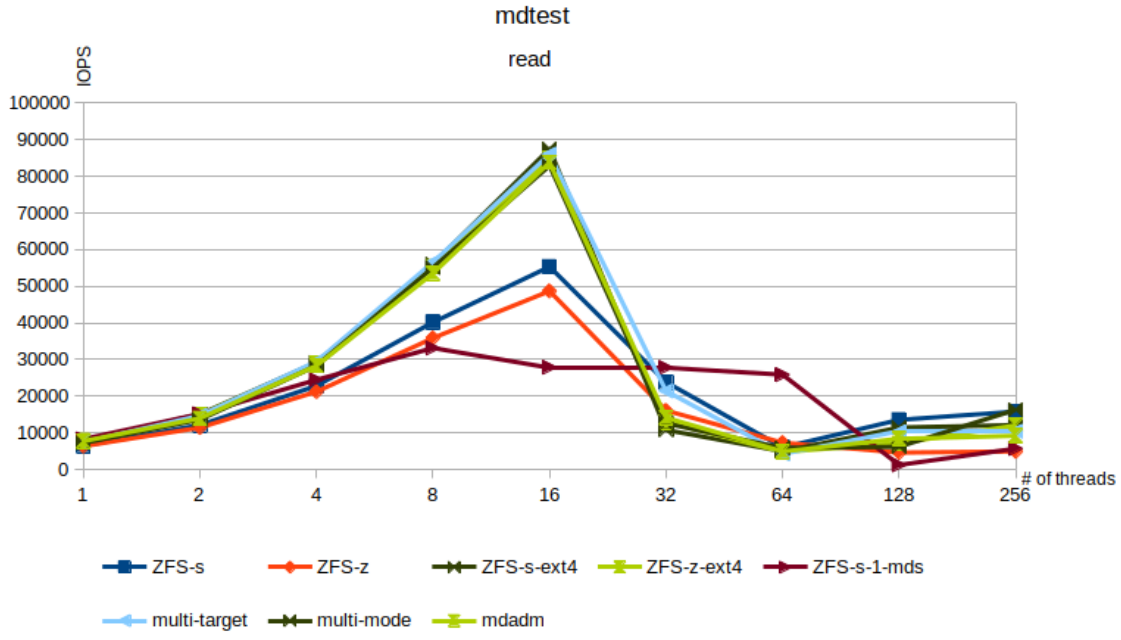


Abbildung 4.13: Metadaten – *File reads* pro Sekunde

Aus den Ergebnissen der Benchmarks mit *mdtest* geht hervor, dass die Leistung der unterschiedlichen Konfigurationen für den Fall *File stats* pro Sekunde ähnlich ist. Die Konfigurationen mit RAID-Z sind ca. 10% langsamer.

Die Konfiguration *ZFS-s-1-mds* mit nur einem Metadata Server zeigt ab mehr als 8 Threads verminderte Leistung.

Die in Abschnitt 4.3.2 festgestellten Leistungseinbußen zeigen auch bei den *mdtest*-Benchmarks ähnliche fallende Leistung bei höherer Anzahl von Threads, die durch die Systemlast der Benchmarks verursacht sein können.

### IOPS-Benchmark

Beim Schreiben von 4 KiB-Blöcken mit zufälligem Offset erzielt die Konfiguration *mdadm* die beste Leistung mit durchschnittlich 475 MiB/s und maximal 878 MiB/s.

Konfigurationen mit ZFS erreichen durchschnittlich 150 MiB/s und maximal 289 MiB/s.

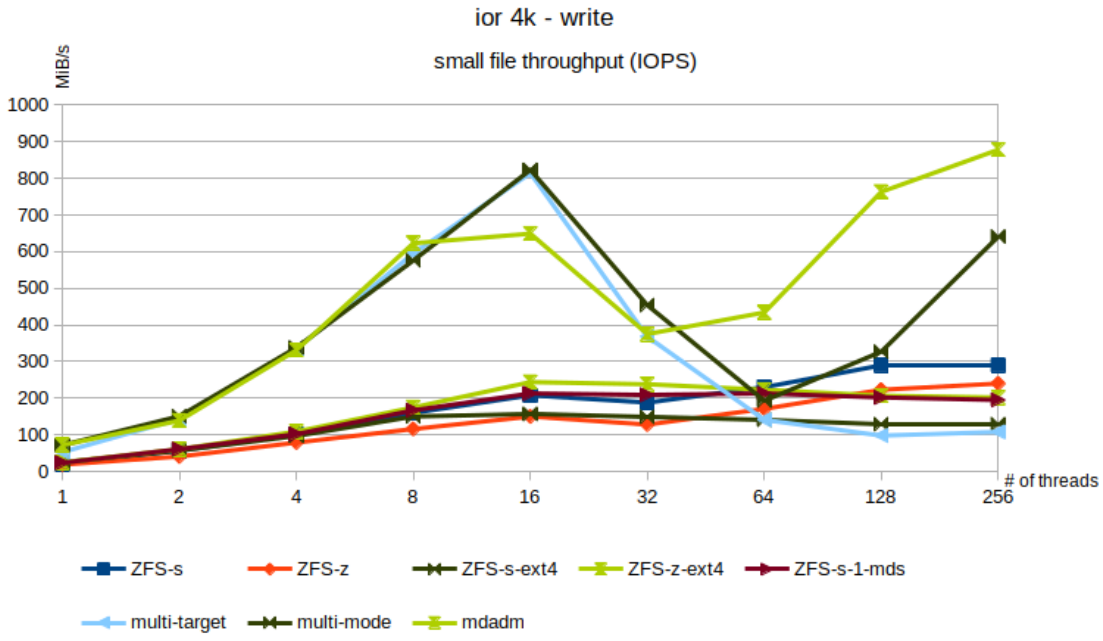


Abbildung 4.14: IOPS (4 KiB) – Schreibvorgang

In diesem Fall entspricht eine Rate von 100 MiB/s 25600 IOPS[1].

Es zeigt sich, dass Konfigurationen mit XFS bessere Schreibleistungen erzielen.

Bei Konfigurationen mit ZFS wird die Schreibleistung kleiner Blöcke auch mit separatem Laufwerk für Metadaten nicht gesteigert.

Der Testfall entspricht einem synthetischen Benchmark im ungünstigsten Fall.

### Zwischenfazit Metadaten-Benchmark

Die Konfigurationen mit XFS zeigen bessere Leistung als ZFS für zufällige und sehr kleine Blockgrößen.

Die Ergebnisse zeigen, dass die Konfigurationen mit ZFS etwa 75% langsamere Metadaten-Leistung erreichen.

Für den Einsatzzweck als verteiltes Dateisystem für Deep Learning Anwendungen ist die verminderte Leistung der Metadata Server mit ZFS jedoch vernachlässigbar.

Sollte das Dateisystem für den Zugriff auf sehr viele kleine Dateien verwendet werden, kann eine Konfiguration mit XFS bzw. separatem Metadaten-Laufwerk besser geeignet sein.

## 4.5 Fazit der Evaluation

Es zeigt sich, dass für das Testszenario *Shared file throughput* Konfigurationen mit ZFS die beste Leistung erzielen und im Vergleich zum bestehenden NFS-Server ein Leistungszuwachs von etwa 1700% beim Lesen und 700% beim Schreiben zu erwarten ist.

Konfigurationen mit ZFS RAID-Z haben im Gegensatz zu Konfigurationen mit ZFS im Striped Modus pro Server ein Paritätslaufwerk. Die Leistung sinkt durchschnittlich um 10% und die Kapazität um 20%. Bei Bedarf kann dadurch Ausfallsicherheit gewonnen werden.

Im Benchmark für Metadaten-Leistung und beim Schreiben von sehr kleinen Blöcken zeigen Konfigurationen mit XFS eine bessere Leistung.

[1] IOPS = MB/s $\times 10^6 \div 4096$

Es wurde festgestellt, dass die Konfiguration mit nur einem Metadata Server gegenüber mehreren eine verminderte Leistung zeigt.

Im Skalierungs-Test für die Konfiguration *ZFS-s* können gute Skalierungs-Eigenschaften festgestellt werden. Verminderte Skalierungseigenschaften, wie bspw. in Diagramm 4.9, bei den Kennlinien *3 Server (1 Client Node)*, können darauf zurückgeführt werden, dass durch weniger ausführende Client-Nodes eine eingeschränkte parallele Last durch die Benchmarks entsteht.

In den Testkonfigurationen agieren Server gleichzeitig als Clients und führen Benchmarks aus. Es wurde in den Durchsatz-Benchmarks (vgl. Abschnitt 4.2) und Metadaten-Benchmarks (vgl. Abschnitt 4.4) eine abfallende Leistung mit steigender Anzahl an Threads festgestellt. In Abschnitt 4.3.2 wurde ermittelt, dass die zusätzliche Systemlast durch Ausführung der Benchmarks nicht unerheblich ist und die Ergebnisse aufgrund dessen abfallende Leistung mit steigender Anzahl an Threads zeigen. Daher sind die Ergebnisse dieser Testreihe nur untereinander vergleichbar.

# 5 Fazit

In diesem Kapitel werden die Ergebnisse der vorherigen Kapitel und der Evaluation zusammengefasst. Es folgen mögliche Ausblicke für die Erweiterung und Optimierung des Systems.

## 5.1 Zusammenfassung

Die stetig steigende Rechenleistung von Prozessoren und GPUs erfordert schnelle Dateisysteme. Es zeigt sich in dieser Arbeit, dass die Konfiguration eines verteilten Dateisystems keine triviale Aufgabe ist. Die optimale Konfiguration des Systems ist stark davon abhängig, für welche Einsatzgebiete das Dateisystem verwendet werden soll.

Ziel der Arbeit war es, ein verteiltes Dateisystem für den Einsatz als Zwischenspeicher für Deep Learning Anwendungen für das Rechenzentrum des FTZ SMSY zu finden und zu konfigurieren. Dabei wurde analysiert, welche Anforderungen relevant sind, welche Systeme diesen gerecht werden und welche Konfiguration des Systems am besten geeignet ist.

In der Analyse (Kapitel 2) wurde der derzeitige Aufbau des Rechenzentrums und die für die Anwendung bereitgestellte Hardware analysiert. Anschließend wurden die Herausforderungen von verteilten Dateisystemen erforscht und anhand einer Anforderungsanalyse wurde ein geeignetes System für den Anwendungsfall ausgewählt. Dabei zeigt sich, dass die fundamentale Herausforderung eines verteilten Dateisystems die entstehenden Latenzen durch die horizontale Skalierung sind.

Im Kapitel 3 wurde dargestellt, wie mit Hilfe von Skripten, die automatisierte Installation und Konfiguration eines Systems umgesetzt wurde. Diese Methode zur Installation und Konfiguration von Systemen ist für Administratoren von wachsenden Rechenzentren

nicht wegzudenken. In großen Rechenzentren mit vielen Servern ist die Wartung und Konfiguration von einzelnen Systemen anderweitig nicht praktikabel.

In der Evaluation (Kapitel 4) zeigen sich erhebliche Leistungsunterschiede der verschiedenen Konfigurationen. Entsprechend der Zielsetzung und gewählter Bewertungskriterien wurde eine Konfiguration mit ZFS als Dateisystem als am besten geeignete Lösung identifiziert. Es ist zu beachten, dass die Systemlast durch das gleichzeitige Ausführen von Benchmarks auf den Servern nicht unerheblich ist.

Als Ergebnis dieser Arbeit zeigt sich, dass das verteilte Dateisystem BeeGFS, mit ZFS als Dateisystem, für den Anwendungsfall als Zwischenspeicher für Compute-Server geeignet ist und einen Leistungszuwachs verspricht. Die entwickelten Automatisierungsskripte können für eine automatisierte Installation des Systems verwendet werden und erleichtern den Wartungsaufwand der Projektgruppe.

In Kapitel 5.2 erfolgt eine detailliertere Analyse der Ergebnisse.

## 5.2 Ergebnisanalyse

In dieser Arbeit wurden für das parallele Dateisystem BeeGFS verschiedene Konfigurationen für das grundlegende Dateisystem verglichen. Das System besteht aus vier Servern mit je fünf NVMe-Laufwerken, die als Storage und Metadata Server fungieren.

Während der Evaluation wurde festgestellt, dass bei der Konfiguration eines parallelen Dateisystems die Wahl des Dateisystems auf den Datenträgern einen großen Einfluss auf die Leistung hat.

Es werden mit dem System auch schon bei geringer Anzahl von Threads (etwa 16) hohe Durchsatz-Geschwindigkeiten erzielt.

Es zeigt sich im *shared file throughput*-Benchmark, dass bei einer Konfiguration mit ZFS als Dateisystem die höchsten Lese- und Schreibgeschwindigkeiten erreicht werden. Für den Einsatz als verteiltes Dateisystem für Deep Learning Anwendungen, wird daher für die vorhandene Konstellation eine Konfiguration der Server mit ZFS als Dateisystem empfohlen.

Die Geschwindigkeiten im Vergleich zum vorhandenen NFS-Server sind im beschriebenen Szenario beim Lesen mit etwa 7800 MiB/s gegenüber 446 MiB/s um den Faktor 17 höher

und beim Schreiben mit etwa 5000 MiB/s gegenüber 672 MiB/s um den Faktor 7 höher. Damit ist bei datenintensiven Anwendungen mit einem erheblichen Leistungszuwachs zu rechnen. Die beobachtete niedrige GPU-Auslastung während datenintensiven Berechnungen, wie in Abschnitt 1.2 dargestellt, sollte minimiert werden und die Berechnungen sollten um ein Vielfaches schneller ausgeführt werden können.

Die Metadaten-Leistung ist bei Konfigurationen mit ZFS gegenüber XFS oder separatem ext4-Laufwerk etwa halbiert.

Beim Schreibvorgang sehr kleiner Blöcke (4 KiB) erreichen Konfigurationen mit XFS um den Faktor 3 höhere Durchsatzgeschwindigkeiten.

Im Skalierungs-Test können gute Ergebnisse festgestellt werden, die allerdings wegen der Systemlast durch das Ausführen der Benchmarks nur begrenzt aussagekräftig sind.

Mit der neuen Konfiguration der NVMe-Server im Rechenzentrum können die Studierenden und Projekte aus dem Bereich Machine Learning viele neue Anwendungsfälle betrachten, zu denen bisher nur erschwert Berechnungen erfolgen konnten.

## 5.3 Ausblick

Mit den Ergebnissen dieser Arbeit, kann die Umsetzung für die Installation und Konfiguration des verteilten Dateisystems auf den Servern des Rechenzentrums erfolgen. Mit der Einbindung des Systems können datenintensive Berechnungen beschleunigt werden. Damit wird eine effizientere Nutzung des Rechenzentrums ermöglicht.

In Zukunft können die Erkenntnisse auch für einen Vergleich mit einem anderen verteilten Dateisystem herangezogen werden.

Die Erkenntnisse dieser Arbeit können für die Erweiterung des Systems weiterverwendet werden, wie im Folgenden genauer beschrieben.

### Erweiterung

Grundsätzlich kann die Erweiterung um weitere Knoten die Leistung erhöhen.

Es ist konzeptionell auch möglich, ähnliche Systeme in weiteren Rechenzentren anderer Standorte zu installieren, die von dem gleichen oder einem anderen Team koordiniert

und verwaltet werden. Durch minimalen Aufwand können die Automatisierungsskripte für weitere Hardwarekonfigurationen angepasst werden.

Wird das System um zusätzliche Server erweitert, kann in Erwägung gezogen werden, das System mit Spiegelservern zu konfigurieren, um das System gegen Ausfälle abzusichern.

Es kann auch eine Systemaufstellung mit heterogener Aufgabenverteilung in Betracht gezogen werden. Dabei kann entsprechend der Ansprüche an das System um zusätzliche Metadata Server oder Storage Server erweitert werden, um die Metadaten-Leistung oder Durchsatz-Leistung zu verbessern.

In jedem Server ist in der aktuellen Konfiguration ein NVMe-Schacht nicht belegt. Um die Metadaten-Leistung zu verbessern, könnte pro Server ein zusätzliches NVMe-Laufwerk (passend für die Konfiguration mit etwa 1 TB Kapazität) installiert werden.

Verschiedene Anbieter bieten eine Softwarelösung für *Block Storage* Dateisysteme. BeeGFS oder andere parallele Dateisysteme können mit einem Software-definierten Dateisystem mit Unterstützung für NVMe-over-Fabric (NVMe-oF) als Varianten konfiguriert werden, die eine bessere Leistung und Latenz für Systeme mit NVMe-Laufwerken versprechen (vgl. [17]). Eine solche Lösung kann in Zukunft als Alternative getestet werden.

Ebenso können weitere Storage Server mit langsameren Laufwerken hinzugefügt werden und Storage Pools konfiguriert werden, die Konfigurationen mit verschiedenen Verzeichnissen und Storage-Geschwindigkeiten ermöglichen (Storage Tiering).

Momentan erfolgt die Installation des Betriebssystems manuell per IPMI Remote-Zugriff und GUI. Um den Aufwand zu zu reduzieren sollte eine Automatisierung der Betriebssystem-Installation in Erwägung gezogen werden, wenn weitere Server zum Rechenzentrum bzw. System hinzugefügt werden. Hierfür kann bspw. ein Installationsdatenträger mit abweichender Konfiguration der Vorgabe für Partitionierung, Einstellungen und Netzwerkkonfiguration erstellt werden.

Das Monitoring des Systems kann über eine zusätzliche Komponente installiert werden und in bspw. Grafana integriert werden (vgl. [49]).

Das Installieren von Updates auf den Systemen kann von den Administratoren mit weiteren Ansible-Playbooks automatisiert werden.

Auch das Entfernen eines Servers aus dem System ist grundsätzlich unproblematisch, wenn das Dateisystem nicht belegt ist (vgl. [50]). Diese Aufgabe kann ebenso mit Ansible-Playbooks automatisiert werden.

Die vorangegangen Erläuterungen zeigen das große Potential für zukünftige Weiterentwicklungen und Verbesserungen. Die Automatisierung der Installation des Systems kann auch genutzt werden, um agil eine Rekonfigurierung des verteilten Dateisystems für individuelle Bedürfnisse vorzunehmen. Sie eröffnet viele neue Möglichkeiten für die Studierenden und ist ein weiterer Aspekt der kontinuierlichen Weiterentwicklung des Rechenzentrums der Forschungsgruppe.

# Literaturverzeichnis


[1] Ieee standard for information technology - portable operating system interfaces (posix(tm)) - part 1: System application program interface (api) - amendment 1: Realtime extension [c language]. *IEEE Std 1003.1b-1993*, pages 1–624, 1994.

[2] AMD. AMD EPYC 9004 SERIES PROCESSORS| AMD DATA SHEET. https://www.amd.com/content/dam/amd/en/documents/products/epyc/epyc-9004-series-processors-data-sheet.pdf, 2023. [Online; accessed 06-December-2023].

[3] Mahsa Bayati, Janki Bhimani, Ronald Lee, and Ningfang Mi. Exploring benefits of nvme ssds for bigdata processing in enterprise data centers. In *2019 5th International Conference on Big Data Computing and Communications (BIGCOM)*, pages 98–106, 2019.

[4] Francieli Boito, Guillaume Pallez, and Luan Teylo. The role of storage target allocation in applications' i/o performance with beegfs. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 267–277, 2022.

[5] Klaus Brüderle. *NVMe statt AHCI*, pages 117–139. Springer Fachmedien Wiesbaden, Wiesbaden, 2022.

[6] Ebubekir BUBER and Banu DIRI. Performance analysis and cpu vs gpu comparison for deep learning. In *2018 6th International Conference on Control Engineering and Information Technology (CEIT)*, pages 1–6, 2018.

[7] Konstantinos Chasapis, Jean-Yves Vet, and Jean-Thomas Acquaviva. Benchmarking parallel file system sensitiveness to i/o patterns. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 427–428, 2019.

[8] Fahim Chowdhury, Yue Zhu, Todd Heer, Saul Paredes, Adam Moody, Robin Goldstone, Kathryn Mohror, and Weikuan Yu. I/o characterization and performance

evaluation of beegfs for deep learning. In *Proceedings of the 48th International Conference on Parallel Processing*, ICPP '19, New York, NY, USA, 2019. Association for Computing Machinery.

[9] Papers With Code. LibriSpeech corpus collection | An ASR corpus based on public domain audio books . https://paperswithcode.com/dataset/librispeech, 2015. [Online; accessed 08-January-2024].

[10] David Cohen, Thomas Talpey, Arkady Kanevsky, Uri Cummings, Michael Krause, Renato Recio, Diego Crupnicoff, Lloyd Dickman, and Paul Grun. Remote direct memory access over the converged enhanced ethernet fabric: Evaluating the options. In *2009 17th IEEE Symposium on High Performance Interconnects*, pages 123–130, 2009.

[11] Super Micro Computer. Supermicro SuperServer | Produktseite. https://www.supermicro.com/products/system/2u/2029/SYS-2029BT-HNR.cfm, 2023. [Online; accessed 18-December-2023].

[12] COCO Consortium. MS COCO Microsoft Common Objects in Context | dataset consisting of 328K images . https://cocodataset.org/. [Online; accessed 08-January-2024].

[13] IBM Corporation. IBM Spectrum Scale license designation | Licensing Agreement . https://www.ibm.com/docs/en/storage-scale/5.0.3?topic=overview-spectrum-scale-license-designation, 2021. [Online; accessed 08-January-2024].

[14] NVIDIA Corporation. ConnectX®-5 EN Card | Mellanox Datasheet. https://network.nvidia.com/files/doc-2020/pb-connectx-5-en-card.pdf, 2020. [Online; accessed 18-December-2023].

[15] NVIDIA Corporation. GPUDirect Storage | A Direct Path Between Storage and GPU Memory. https://developer.nvidia.com/blog/gpudirect-storage/, 2023. [Online; accessed 18-December-2023].

[16] Samsung Electronics. Product Brief Samsung PM983 | Samsung PM983 NF1 NVMe SSD Datasheet. https://download.semiconductor.samsung.com/resources/data-sheet/Product_Brief_Samsung_PM983_NF1_NVMe_SSD_1806.pdf, 2019. [Online; accessed 18-December-2023].

[17] Excelero. NVMesh SUPERCHARGES BeeGFS | Joint Solution BeeGFS Excelero . https://www.excelero.com/wp-content/uploads/2019/09/NVMesh-supercharges-BeeGFS.pdf, 2023. [Online; accessed 26-December-2023].

[18] FTZ. FTZ SMSY Buchungssystem| Buchung der Server. https://mrbs.csti.haw-hamburg.de/, 2023. [Online; accessed 06-December-2023].

[19] Kyu J. Han, Ramon Prieto, Kaixing Wu, and Tao Ma. State-of-the-art speech recognition using multi-stream self-attention with dilated 1d convolutions, 2019.

[20] Thomas Handschuch. *Verteilte Dateisysteme —NFS*, pages 474–507. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[21] Jan Heichler. Evaluating the MetaData Performance of BeeGFS® | Performance of different backend filesystems . https://www.beegfs.io/docs/whitepapers/Metadata_Performance_Evaluation_of_BeeGFS_by_ThinkParQ.pdf, 2015. [Online; accessed 09-December-2023].

[22] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[23] Fraunhofer IMW. MASCHINELLES LERNEN | EINE ANALYSE ZU KOMPETENZEN, FORSCHUNG UND ANWENDUNG. https://www.bigdata-ai.fraunhofer.de/content/dam/bigdata/de/documents/Publikationen/Fraunhofer_Studie_ML_201809.pdf, 2018. [Online; accessed 06-December-2023].

[24] Google Inc. AI and machine learning products | Products, solutions, and services. https://cloud.google.com/products/ai?hl=en, 2023. [Online; accessed 06-December-2023].

[25] Kristina Kaličanin Bojić, Milica Čolović, Angelina Njeguš, and Vladimir Mitic. Benefits of artificial intelligence and machine learning in marketing. pages 472–477, 04 2019.

[26] The kernel development community. ext4 Data Structures and Algorithms | 4.1.1. Inode Size . https://www.kernel.org/doc/html/v5.0/filesystems/ext4/dynamic.html. [Online; The Linux Kernel 5.0.0; accessed 02-January-2024].

[27] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima, 2017.

[28] Ronny Krashinsky. Ampere architecture in depth | Nvidia Data Science Technical Blog. https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/, 2020. [Online; accessed 06-December-2023; Release: May 14th 2020].

[29] Michael Kuhn. Parallele dateisysteme. *Informatik Spektrum*, 42(5):360–364, 2019.

[30] William Loewe, Tyce McLarty, and Christopher Morrone. HPC IO Benchmark Repository. https://github.com/hpc/ior, 2023. [Online; accessed 14-August-2023].

[31] University of Central Florida. UCF101 | Action Recognition Data Set . https://www.crcv.ucf.edu/data/UCF101.php, 2013. [Online; accessed 08-January-2024].

[32] Regents of the University of California. IOR Options | Command line options . https://ior.readthedocs.io/en/latest/userDoc/options.html, 2018. [Online; accessed 08-January-2024].

[33] Inc. O'Reilly Media. Understanding the Linux Kernel, 3rd Edition | The Page Cache . https://www.oreilly.com/library/view/understanding-the-linux/0596005652/ch15s01.html. [Online; accessed 02-January-2024].

[34] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, 2015.

[35] Prometeus. List Statistics | TOP500. https://www.top500.org/statistics/list/, 2023. [Online; accessed 03-October-2023; Query Release: June 2023, Category: Operating system Family].

[36] Nishanth S, Manu S Rao, Sagar B M, Padmashree T, and Cauvery N K. Performance of cpus and gpus on deep learning models for heterogeneous datasets. In *2022 6th International Conference on Electronics, Communication and Aerospace Technology*, pages 978–985, 2022.

[37] Nikita Sharma, Ruihao Li, Qinzhe Wu, and Lizy K. John. Performance impact of nvme-over-tcp on hdfs workloads. In *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*, pages 334–339, 2022.

[38] Yongseok Son, Hara Kang, Hyuck Han, and Heon Young Yeom. An empirical evaluation and analysis of the performance of nvm express solid state drive. *Cluster Computing*, 19(3):1541–1553, 2016.

[39] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild, 2012.

[40] Open Speech and Language Resources. LibriSpeech ASR corpus | Large-scale (1000 hours) corpus of read English speech . https://www.openslr.org/12, 2015. [Online; accessed 08-January-2024].

[41] Andrew S. Tanenbaum and Herbert Bos. *Moderne Betriebssysteme* . Pearson Deutschland, 2016.

[42] Andrew S. Tanenbaum and Maarten van Steen. *Verteilte Systeme Prinzipien und Paradigmen*. Pearson Deutschland, 2007.

[43] ThinkParQ and Fraunhofer ITWM. BeeGFS Architecture | Overview . https://doc.beegfs.io/latest/architecture/overview.html, 2023. [Online; accessed 08-January-2024].

[44] ThinkParQ and Fraunhofer ITWM. BeeGFS EULA | END USER LICENSE AGREEMENT (Server) . https://www.beegfs.io/docs/BeeGFS_EULA.txt, 2023. [Online; accessed 08-January-2024].

[45] ThinkParQ and Fraunhofer ITWM. Benchmarking a BeeGFS System | External Benchmarking Tools . https://doc.beegfs.io/latest/advanced_topics/benchmark.html#external-benchmarking-tools, 2023. [Online; accessed 21-December-2023].

[46] ThinkParQ and Fraunhofer ITWM. Client Side Caching Modes | Client Caching . https://doc.beegfs.io/latest/advanced_topics/client_caching.html, 2023. [Online; accessed 26-December-2023].

[47] ThinkParQ and Fraunhofer ITWM. Filesystem Modification Events | Configuration of Filesystem Modification . https://doc.beegfs.io/latest/advanced_topics/filesystem_modification_events.html, 2023. [Online; accessed 08-January-2024].

[48] ThinkParQ and Fraunhofer ITWM. Metadata Node Tuning | File System recommendations. https://doc.beegfs.io/latest/advanced_topics/metadata_tuning.html, 2023. [Online; accessed 09-December-2023].

[49] ThinkParQ and Fraunhofer ITWM. Monitoring Service | Monitoring installation . https://doc.beegfs.io/latest/advanced_topics/mon.html, 2023. [Online; accessed 26-December-2023].

[50] ThinkParQ and Fraunhofer ITWM. Node Management | Removing Nodes . https://doc.beegfs.io/latest/advanced_topics/node_mamagement.html#removing-nodes, 2023. [Online; accessed 26-December-2023].

[51] ThinkParQ and Fraunhofer ITWM. Quick Start Guide | Installation von BeeGFS . https://doc.beegfs.io/latest/quick_start_guide/quick_start_guide.html, 2023. [Online; accessed 21-December-2023].

[52] ThinkParQ and Fraunhofer ITWM. Store Node Tuning | File System recommendations. https://doc.beegfs.io/latest/advanced_topics/storage_tuning.html, 2023. [Online; accessed 09-December-2023].

[53] ThinkParQ and Fraunhofer ITWM. System Requirements | Disk Space Requirements. https://doc.beegfs.io/latest/system_design/system_requirements.html, 2023. [Online; accessed 09-December-2023].

[54] Tao Wang, Shihong Yao, Zhengquan Xu, Lian Xiong, Xin Gu, and Xiping Yang. An effective strategy for improving small file problem in distributed file system. In *2015 2nd International Conference on Information Science and Control Engineering*, pages 122–126, 2015.

[55] Tiezhu Zhao and Jinlong Hu. Performance evaluation of parallel file system based on lustre and grey theory. In *2010 Ninth International Conference on Grid and Cloud Computing*, pages 118–123, 2010.

# A Anhang

Im Folgenden sind weitere Diagramme zu Abschnitt 4.3.2 zu finden.

## A.1 Diagramme

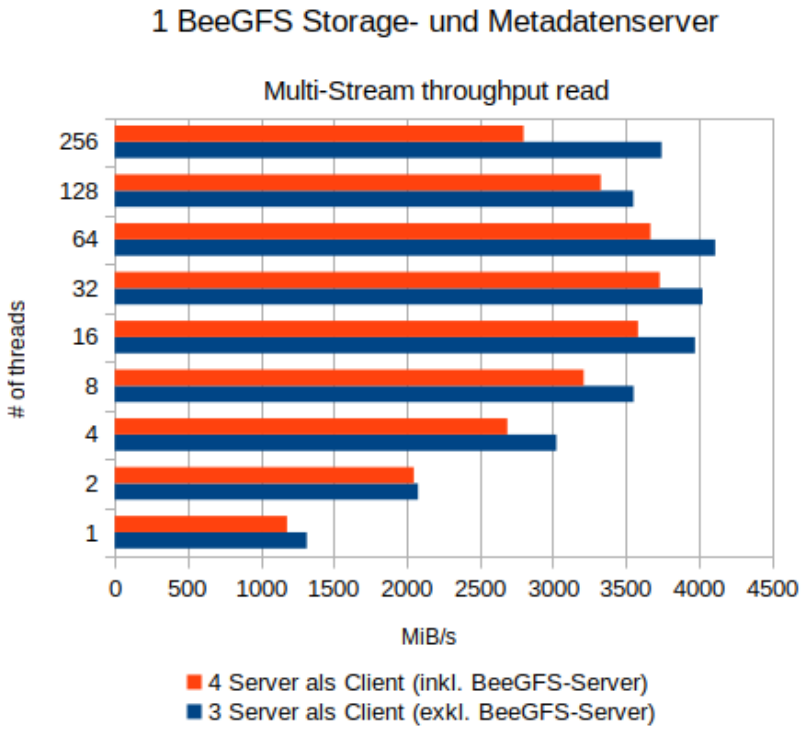


Abbildung A.1: Systemlast 1 Server ZFS striped (2 MB) – Lesevorgang

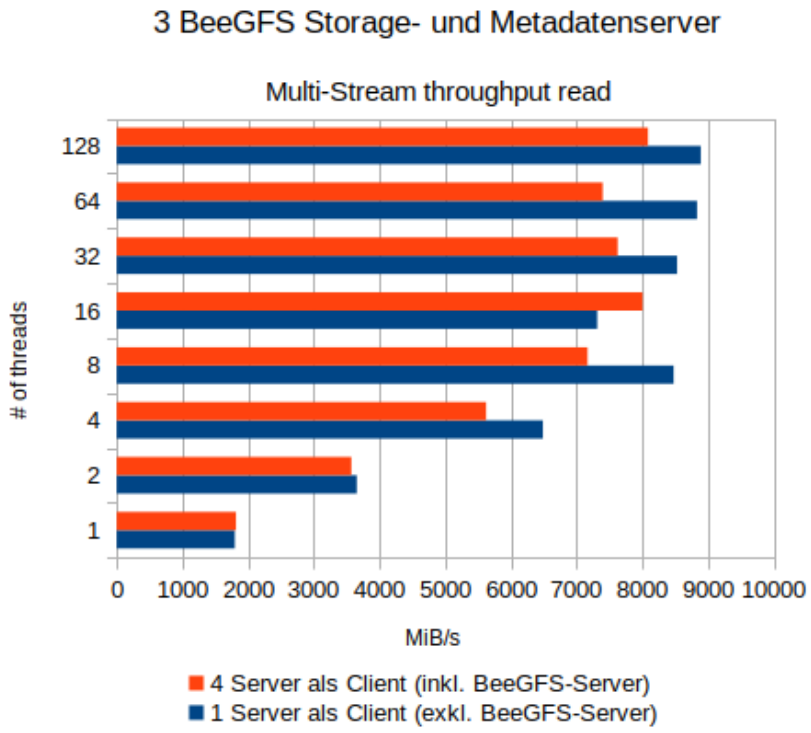


Abbildung A.2: Systemlast 3 Server ZFS striped (2 MB) – Lesevorgang

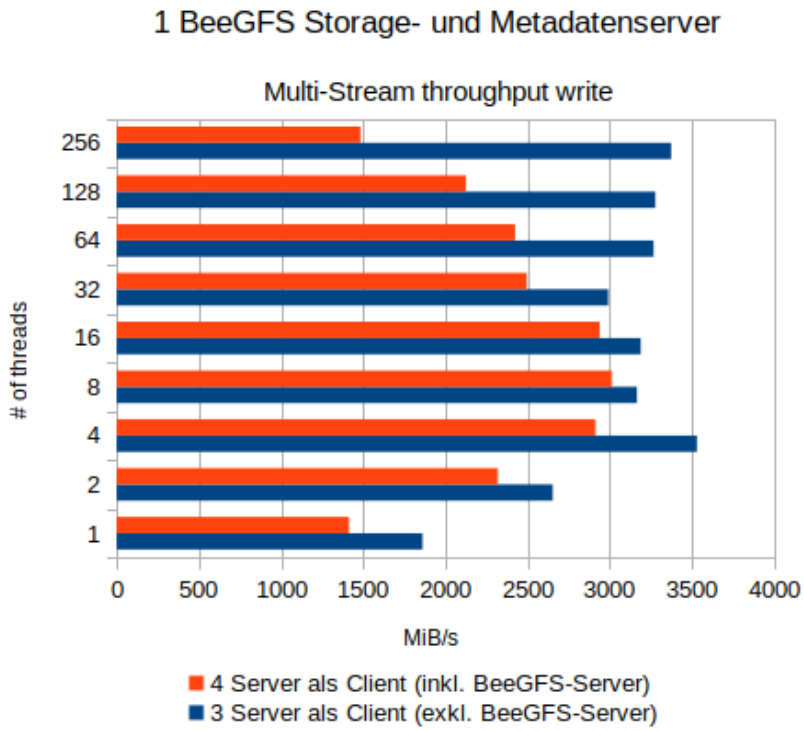


Abbildung A.3: Systemlast 1 Server ZFS striped (2 MB) – Schreibvorgang

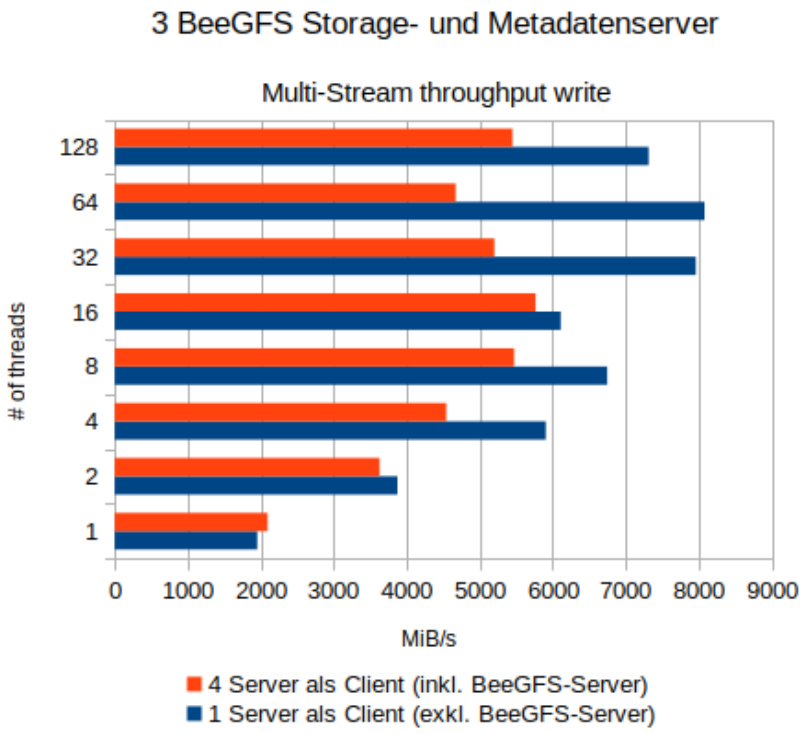


Abbildung A.4: Systemlast 3 Server ZFS striped (2 MB) – Schreibvorgang

## A.2 CPU-Auslastung

Das folgende Diagramm zeigt die CPU-Auslastung während der Ausführung des *Multi-stream throughput*-Benchmark mit 16 Threads (verteilt über 4 Server).

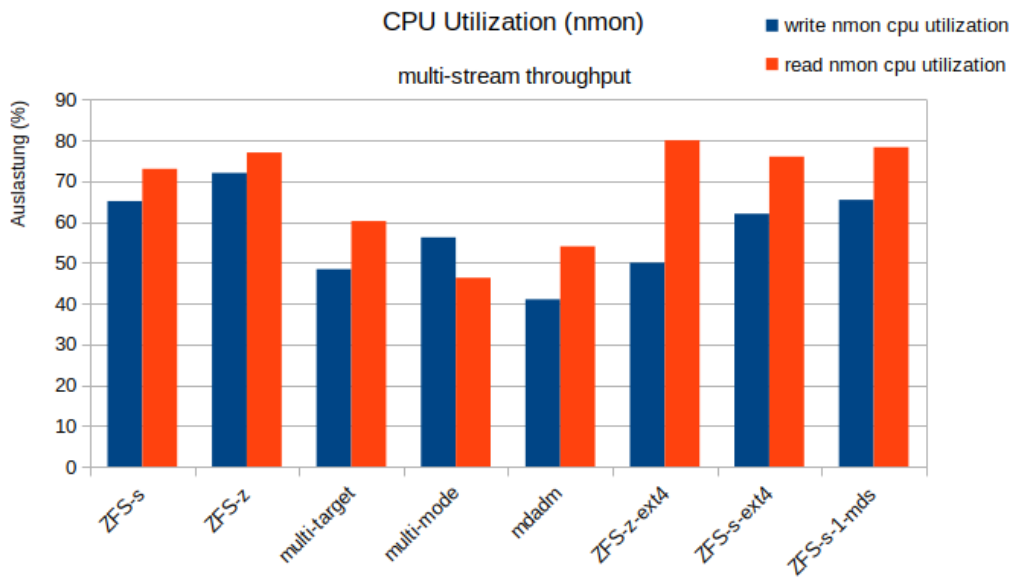


Abbildung A.5: CPU-Auslastung mit 16 Threads – nmon

## A.3 Readme - Stand Januar 2024

Folgendes Dokument wurde als Leitfaden für die Benutzung der Automatisierungsskripte erstellt. Es berücksichtigt eine Version, in der ZFS als Striped oder RAID-Z konfiguriert werden kann.

# B Installation von BeeGFS auf den vier NVME-Nodes

Aktuell ist die Installation von `BeeGFS 7.4.0` unter `AlmaLinux 8.8` bis Kernel `4.18.0-477.13.1.el8_8.x86_64` getestet, kompatibel mit den Playbooks und funktionsfähig.

## B.1 Installation des Betriebssystems

1. Die Installation über das Remote-Management-Interface der Server beim Boot via **F12** (PXE-Boot) mit `"almaLinux8"` Eintrag, starten oder **F2** falls per JAVA-Plugin ein ISO-Image eingehängt wird. Nach einer weile muss man ggf. die Meldung `"Driver disk device selection"` mit `c` bestätigen.
2. Sprachauswahl bestätigen und ggf. Keyboard-Layout anpassen
3. In dem Installationsmenü einen Benutzer erstellen (als Admin-Benutzer).
4. Als Installations-Quelle den nähsten Server auswählen (*Closest mirror*) und als Software-Auswahl `"minimal"`.
5. Als Zielfestplatten für das System die beiden kleinen NVMe-SSD's (*nvme0n1* und *nvme1n1*) markieren und ggf. Speicherplatz freigeben (*reclaim space* > *delete all*)
6. In der Netzwerkkonfiguration den Hostnamen eintragen und anwenden
7. Da der SSH-Zugriff nur per VLAN-346 möglich ist, muss während der Installation entsprechend ein Bond und VLAN erstellt werden.

**Netzwerkkonfiguration**

- Schnittstellen *ens6f0* und *ens6f1* konfigurieren: unter Allgemein/General `"automatische Verbindung mit Priorität"` haken entfernen.
- Einen Bond erstellen mit: Modus *802.3ad* und *ens6f0* und *ens6f1* als Slaves/Ports, MAC-Adresse der Ports jeweils manuell konfigurieren (Tabelle), IPV4-Konfiguration

(*disabled*) und IPV6- (*ignored*). (HINWEIS für almaLinux9-Installer: wenn das zweite Fenster über einem Fenster geöffnet wird (Bond-Ports hinzufügen), kann man es nicht auswählen, wenn das alte Fenster im Hintergrund ist - erst das untere zur Seite schieben)

- VLAN-Interface erstellen, mit dem eben erstellten Bond als Port und VLAN-Kennung *346* konfigurieren
- Nach kurzer Zeit erhält der NVMe-Server vom DHCP-Server die korrekte IP.

| Hostname | MAC-Adresse | IP-Adresse |
|---|---|---|
| nvme01a | 76:b5:83:80:b2:ba | 10.0.0.184 |
| nvme01b | 76:b5:83:80:b2:bb | 10.0.0.185 |
| nvme01c | 76:b5:83:80:b2:bc | 10.0.0.186 |
| nvme01d | 76:b5:83:80:b2:bd | 10.0.0.187 |

8. Die Installation kann jetzt gestartet werden und anschließend der Neustart nach der Installation ausgeführt werden.

## B.2 Nach der Installation des Betriebssystems

Warten bis die Server per SSH erreichbar sind.

Falls die Server nicht erreichbar werden, kann es sein, dass das VLAN nicht aktiv ist. In dem Fall muss per Management-Zugriff mit *nmtui* ggfs. ens6f0 und ens6f1 *deaktiviert* und das VLAN *aktiviert* werden.

Wenn noch kein Key vorhanden/ein neuer erstellt wird: `ssh-keygen -t ed25519 -C ‘„ansible_nvme‘„`

Und entsprechend einen *Key* mit der Bezeichnung "ansible" erstellen. *Der Inhalt der ansible.pub muss dann in der bootstrap.yml ersetzt werden.*

Es muss der ssh-key auf den Server kopiert werden (Benutzername des Server-Benutzers mit Root-Rechten - ist der Benutzername, der bei der Installation gewählt wurde)

```
ssh-copy-id -i ~/.ssh/ansible.pub {username@}[server-ip]
```

Bootstrappen ("username" ist wieder der bei der Installation gewählte)

```
ansible-playbook bootstrap.yml { -limit [server-ip] } -ask-become-pass -u <username>
```

Standardmäßig wird ein ZFS-Pool im Striped-Modus erstellt. Die Variable `raidz: false` kann auf `raidz: true` geändert weren, um raidz (raid5) für den ZFS-Pool zu konfigurieren.

Zum *installieren* von BeeGFS

```
ansible-playbook install_beegfs.yml { -limit [server-ip] }
```

## B.3 Inventory

Die verschiedenen Rollen des Playbooks sind aktuell auf die Hauptdienste aufgeteilt.

*(Hinweis: Damit die mirrorgroup-Konfiguration auf dem managment-node ausgeführt werden kann, muss dieser gleichzeitig als client-host eingetragen sein...)*

## B.4 Installation des BeeGFS-Clients

Sollen außer den vorhandenen vier Nodes weitere Server installiert werden, müssen Sie entsprechend in der Inventory-Datei eingetragen werden.

Es muss im Verzeichnis `host_vars/` eine YAML-Datei angelegt werden.

- Die Angabe der Variable `management_node: [hostname-management-node]` ist erforderlich. (Standardmäßig ist es also `management_node: nvme01a`.)
- Die Variable zur Erkennung der Distributionsversion ist ebenso erforderlich: `dist-tag: .el{{ ansible_distribution_major_version }}` *Funktionstüchtig ist hier momentan auch nur AlmaLinux 8 - für andere Distributionen müsste die Installation getestet und angepasst werden*

Mit `ansible-playbook install_client.yml { -limit [server-ip] }` kann der BeeGFS-Client auf zusätzlichen Nodes installiert werden.

## B.5 Konfigurieren von Mirrorgroups

Zum konfigurieren der Mirror-Groups muss das `configure_mirrorgroups.yml` Playbook von einem Client gestartet werden.

```
ansible-playbook configure_mirrorgroups.yml -limit \[server-ip\]
```

*Die Konfiguration ist "hardcoded", so dass Node A und Node B gespiegelt sind und Node C und Node D gespiegelt sind.*

## B.6 Starten/Stoppen der Dienste

Zum Stoppen oder Starten aller BeeGFS-Dienste können die Playbooks `start_services.yml` oder `stop_services.yml` verwendet werden.

## B.7 Entfernen von Nodes

Soll ein Node aus der Konfiguration entfernt werden, ist es am besten, wenn das BeeGFS-Verzeichnis geleert wird (`rm -rf /mnt/beegfs/*`).

Zum Löschen eines Nodes, entsprechend der Dokumentation BeeGFS-Dokumentation (Node Management)[1]

```
find /data/beegfs/meta/{dentries,inodes,buddymir/dentries,buddymir/inodes}
-type f
```

**Metadaten-Node**

Sollte

```
/data/beegfs/meta/inodes/38/51/root\ /data/beegfs/meta/inodes/35/5B/disposal\
/data/beegfs/meta/buddymir/inodes/23/40/disposal\
```

zurückgeben - dann ist das Metadaten-Verzeichnis leer.

Und der Metadaten-Node kann mit `beegfs-ctl -removenode -nodetype=meta {node-id}` entfernt werden.

[1] <https://doc.beegfs.io/latest/advanced_topics/node_mamagement.html>

### Storage-Node

Zum migrieren der Daten von einem Storage-Node, kann mit `beegfs-ctl -migrate -nodeid={node-id} /mnt/beegfs` der Inhalt auf die übrigen Nodes verschoben werden.

Mit `beegfs-ctl -removenode -nodetype=storage {node-id}` kann der Storage-Node entfernt werden.

## Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

| Ort | Datum | Unterschrift im Original |
| --- | --- | --- |