Schlussbericht

Zuwendungsempfänger:	Förderkennzeichen:
Georg-August-Universität Göttingen	16ME0663K
Technische Universität Dresden	
Fraunhofer ITWM	
Dt. Zentrum für Luft- und Raumfahrt e.V.	

Vorhabenbezeichnung:

Skalierbare und performante Massenspeicherzugriffe für Exascale-Supercomputer – MCSE

Laufzeit des Vorhabens: 01.10.2022-31.12.2025

Autoren:
Patrick Höhn (GAUG), Michael Kluge (TUD), Philipp Falk (ThinkParQ), Dominik Vietinghoff (DLR), Philipp Zimmermann (ITWM)

SPONSORED BY THE

Federal Ministry
of Education
and Research

Funded by
the European Union
NextGenerationEU

PROJEKTBETEILIGTE

Verbundkoordinator	Name	Georg-August-Universität Göttingen
	Adresse	Goßlerstr. 5-7, 37073 Göttingen
	Abkürzung	GAUG
Ansprechpartner	Name	Julian Kunkel
	Telefon	0551/39-30144
	Email	julian.kunkel@gwdg.de
Unterauftragnehmer:		
ThinkParQ GmbH		Philipp Falk
Trippstadterstrasse 113,		philipp.falk@thinkparq.com
67663 Kaiserslautern		Tel. 0631 277576300

Adressen und Ansprechpartner der Verbundpartner

Institution	Kürzel	Ansprechpartner
Technische Universität Dresden TU Dresden, 01062 Dresden	TUD	Prof. W.E. Nagel, wolfgang.nagel@tu-dresden.de Tel. 0351 463-35450
Fraunhofer ITWM Fraunhoferplatz 1, 67663 Kaiserslautern	ITWM	Rui Machado, rui.machado@itwm.fraunhofer.de Tel. 0631 31600-4392
Dt. Zentrum für Luft- und Raumfahrt e.V. Zwickauer Straße 46, 01069 Dresden	DLR	Thomas Gerhold, thomas.gerhold@dlr.de Tel. 0351 21071- 103

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Kurzdarstellung

1.1 Aufgabenstellung

Exascale-Systeme waren bei Antragsstellung im Jahr 2021 für die nahe Zukunft erwartete HPC-Systeme. Das erste bekannte Exascale-System ist dabei das im Juni 2022 bei der Verkündung der TOP500 auf der ISC-Konferenz der Frontiers-Supercomputer in den USA. Selbst im Jahr 2026 existiert nur eine überschaubare Anzahl von Exacscale-Systemen, wobei im Jahr 2025 Europa mit dem Jupiter System am Jülich Supercompute Center sein erstes Exascale-System erfolgreich installiert hat. Trotz der beachtlichen Rechenleistung dieser Rechencluster hat sich die verfügbare Speicherbandbreite nicht in gleichem Maße weiterentwickelt, wodurch die Leistung der Systeme in wachsendem Ausmaß von der Geschwindigkeit der Datentransfers abhängen.

Im Rahmen des MCSE-Projekts werden deshalb Ansätze für skalierbare und performante Massenspeicherzugriffe für Exascale-Supercomputer untersucht und Verbesserungen vorgeschlagen. Dabei müssen nicht nur auf die klassischen HPC-Anwendungen mit ihren typischen schreibintensiven Zugriffsmustern, sondern auch die neueren Anwendungen aus dem Bereich der künstlichen Intelligenz mit dem Lesezugriff auf eine sehr große Anzahl von kleinen Dateien eingegangen werden. Ein grundlegendes Problem bestehender Systeme ist dabei die semantische Lücke zwischen einer Vielzahl von vorhandenen High-Level-Bibliotheken in den oberen Schichten des I/O-Stacks und tiefen Schichten bis hin zur zugrundeliegenden Hardware. Dabei können die Absichten der Anwendung bei der Optimierung nicht berücksichtigt werden, da die reichhaltigen Semantiken in den High-Level-Bibliothken nicht an die unteren Schichten durchgereicht werden können. Es ist Aufgabe des vorliegenden Projekts diese Lücke zu schließen und dadurch Speichersystem optimal für die Anforderungen in verschiedenen Phasen von Simulationsprojekten zu nutzen.

Darüber hinaus bestehen Lücken in vorhandenen Simulationscodes, wie z. B. dem am DLR entwickelten CODA, die der effizienten Skalierung von Speicherzugriffen entgegen stehen. Simulationscodes sollten durch die explizite Deklaration von Anforderungen in Hinblick auf Zugriffssemantiken, die Anpassung der Speichersystem auf Basis echter Anforderungen und nicht nur wie bisher üblich auf Basis von Worst-Case-Annahmen ermöglichen. Dadurch sollte sich die Simulationsgeschwindigkeit dieser Codes erheblich beschleunigen lassen.

1.2 Voraussetzungen, unter denen das Vorhaben durchgeführt wurde

Georg-August-Universität Göttingen (GAUG)

Die Universität Göttingen ist eine international bedeutende Forschungsuniversität. Sie deckt mit 13 Fakultäten ein differenziertes Fächerspektrum in den Natur-, Geistes- und Sozialwissenschaften sowie der Medizin ab und bietet durch das Umfeld von acht außeruniversitären Forschungsinstituten am Göttingen Campus disziplinübergreifende Zusammenarbeit. So existieren mit dem Campus-Institut Data Science (CIDAS[1]) und dem Simulationswissenschaftlichen Zentrum (SWZ[2]) Kompetenzzentren, die Forschung zu Grundlagen aber auch mit Anwendungsbezug betreiben. Die Universität Göttingen arbeitet eng mit der GWDG zusammen. An der Universität wird Spitzenforschung u.a. in den Bereichen der Medizin, Lebenswissenschaften und Digital Humanities durchgeführt, welche massive Datenmengen verarbeiten.

Prof. Kunkel forscht seit vielen Jahren im Bereich datenintensive Wissenschaft und der Hochleistungs-Ein-/Ausgabe. Prof. Kunkel ist Mitbegründer der IO500[3], ist am Centre of Excellence in Simulation of Weather and Climate in Europe (ESiWACE[4]), einem H2020 geförderten Projekt, beteiligt und entwickelt bspw. mit ESDM[5] eine Middleware für large-scale I/O. Für die HPC Community organisiert er regelmäßig BoF Sessions auf u.a. der ISC-HPC Konferenz zum Thema Datenmanagement. Über seine Ein-/Ausgabe- und Datenmanagment-bezogenen Tätigkeiten hinaus ist er auch an Forschung zur Steigerung der Effizienz und Kosten-Nutzenberechnung an Rechenzentren involviert.

Im Projekt wird die Universität die theoretische Analyse von IOVerbs leiten und die Koordination übernehmen. Die Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen (GWDG[6]) ist ein gemeinsames

[1] CIDAS: `https://www.uni-goettingen.de/de/634473.html`
[2] SWZ: `https://www.simzentrum.de/en/`
[3] IO500: `https://io500.org/`
[4] ESiWACE `https://www.esiwace.eu/`
[5] ESDM: `https://github.com/ESiWACE/esdm`
[6] GWDG: `https://www.gwdg.de/`

Rechenzentrum und Forschungseinrichtung der Max-Planck Gesellschaft und der Universität Göttingen und wird das Projekt unterstützen. Als nationaler Tier 2-Standort im NHR bietet sie ein Portfolio von Beratung und Unterstützung im Bereich des Hochleistungsrechnens.

Technische Universität Dresden (TUD)

Die TU Dresden ist eine der größten Technischen Universitäten und eine der führenden und dynamischsten Hochschulen in Deutschland. Seit 2012 gehört sie zu den deutschen Exzellenz-Universitäten. Das Zentrum für Informationsdienste und Hochleistungsrechnen (ZIH) der TU Dresden ist seit vielen Jahren das Kompetenzzentrum für Hochleistungsrechnen für die TU Dresden und alle Hochschulen und akademischen Forschungseinrichtungen in Sachsen. Seit Anfang 2021 ist es eines von neun Zentren des NHR-Verbunds. Zu den methodischen Schwerpunktthemen des NHR-Zentrums am ZIH gehören „High Performance Data Analytics", „Innovative Storage-Architekturen" und „Performance- und Energieeffizienz-Analyse und Optimierung", die direkt mit dem vorliegenden Antrag korrespondieren. Das ZIH der TU Dresden forscht schon lange im Bereich der Performance-Analyse und entwickelt aktiv an Performance-Analyse Werkzeugen wie Vampir[7], Score-P[8] oder lo2s[9], die neben klassischer Performance-Analyse insbesondere auch die Analyse des E/A-Verhaltens von Anwendungen unterstützen. Weiterhin werden am ZIH Speichertechnologien und alternative Dateisysteme erforscht um neuartige Data Analytics und datenintensive Workflows bestmöglich zu unterstützen. Die TU Dresden legt ihren inhaltlichen Schwerpunkt im Projekt auf die Spezifikation und Implementierung der IOVerbs sowie der Abbildung der POSIX-API auf die IOVerbs. Als NHR-Zentrum stellt es außerdem HPC- und Storage-Ressourcen bereit, die für besonders datenintensive HPC-Workloads ausgelegt sind.

Fraunhofer ITWM

Als Teil des Fraunhofer ITWM entwickelt das Competence Center for High-Performance Computing (CC-HPC) innovative HPC-Lösungen für die Industrie und beteiligt sich an nationalen und internationalen Forschungsprogrammen. Seit seiner Gründung im Jahr 2002 konzentriert sich das CC-HPC vor allem auf die Entwicklung paralleler Anwendungen und auf die Entwicklung von HPC-Werkzeugen. Dazu gehören die Kommunikationsbibliothek GPI (Global Address Space Programming Interface) und die Task basierte Programmierumgebung GPI-Space.[10] GPI-Space erlaubt es Entwicklern Domain spezifische Workflow-Systeme zu entwickeln, die anwendungsspezifische Parallelisierungspattern, Daten Management Strategien und I/O Pattern enthalten. Das GPI-Space System nutzt dabei eine Petri-Netz basierte Workflow Engine um Abhängigkeiten zu modellieren. Das Infinite Memory Layer (IML) ist ein in das GPI-Space System integriertes verteiltes I/O Subsystem, welche die in Memory Daten verwaltet und automatisierte Datentransfers managed. Sowohl das Seismic Processing System ALOMA als auch das führende Computer Algebra System Singular nutzen zur Parallelisierung, zum Datenmanagement und Workflow Orchestrierung das GPI-Space System mit dem integrierten IML Layer. Das CC-HPC hat das parallele BeeGFS Dateisystem entwickelt und arbeitet heute eng mit der Firma ThinkParQ (ITWM Spin-Off) zusammen. Im Rahmen von MCSE wird das IML System zum MCS2 System weiterentwickelt, welches die IOVerbs Spezifikation unterstützen wird.

DLR

Das DLR ist das Forschungszentrum der Bundesrepublik Deutschland für Luft- und Raumfahrt. Es betreibt Forschung und Entwicklung in Luftfahrt, Raumfahrt, Energie und Verkehr, Sicherheit und Digitalisierung. An 30 Standorten beschäftigt das DLR circa 9.000 Mitarbeiter. Das Institut für Softwaremethoden zur Produkt-Virtualisierung in Dresden befasst sich mit der Erforschung und Entwicklung von informatisch/technischen Grundlagen zur Beschreibung und Realisierung des virtuellen Produkts in der Luftfahrt auf Basis höherwertiger, multidisziplinärer Simulationsverfahren. Die Abteilung für Hochleistungsrechnen entwickelt hochskalierbare Simulationssoftware zur effizienten Nutzung aktueller und zukünftiger Hochleistungsrechner-Architekturen. Performance-Analyse und -Optimierung sowie Skalierbarkeitsuntersuchungen der verwendeten Simulationssoftware bilden einen weiteren thematischen Schwerpunkt. Als Kompetenzzentrum wird zusammen mit anderen Instituten und Industriepartnern unter anderem der Strömungslöser CODA (CFD for Onera, DLR, and Airbus) entwickelt, der in das parallele Kopplungsframework

[7] Vampir: `https://vampir.eu/`
[8] Score-P: `https://www.vi-hps.org/projects/score-p`
[9] lo2s: `https://github.com/tud-zih-energy/lo2s`
[10] GPI-Space: `http://www.gpi-space.de`

FlowSimulator Data Manager (FSDM) für multidisziplinäre Simulationen eingebunden ist. Sowohl die Simulationsplattform FSDM als auch der Strömungslöser CODA werden bereits in mehreren Luftfahrtunternehmen produktiv genutzt. Das DLR bringt seine Expertise in den genannten Softwarekomponenten in das Projekt ein und entwickelt Erweiterungen am FSDM Framework, um die Skalierbarkeit der E/A-Operationen zu erhöhen.

1.2.1 Assozierte Partner

ThinkParQ Die ThinkParQ GmbH ist ein mittelständisches Unternehmen mit Firmensitz in Kaiserslautern, welches sich mit der Entwicklung, dem technischen Support und Vertrieb des parallelen Filesystems BeeGFS im Bereich High-Performance Computing beschäftigt. Die Software steht als Open Source der Community weltweit zur Nutzung zur Verfügung und wird in vielen HPC-Systemen erfolgreich eingesetzt. Der Vertrieb der Lösung erfolgt durch weltweite Vertriebspartner, ThinkParQ beschäftigt ca. 20 Mitarbeiter. Im Rahmen des MCSE Projektes beabsichtigt ThinkParQ sich mit der IOVerbs Integration, den neuen Konzepten zu den IOVerbs und MCS2 konzeptionell und in der Entwicklung auseinanderzusetzen.

1.3 Planung und Ablauf des Vorhabens

Unser Ziel in diesem Vorhaben war es die Trennung von Arbeitsspeicher und nichtflüchtigem Speicher aufzuheben und in einem verteilten Computersystem unter einer einheitlichen Schnittstelle zusammenzubringen, dass verschiedene Klassen von Speichermedien flexibel in HPC Workflows nutzbar macht. Diese Vision wurde mittels zwei sich ergänzenden Ansätzen umgesetzt:

1) Zunächst fand die systematische Untersuchung von alternativen Schnittstellen für elementare Ein-/Ausgabe (E/A) von parallelen Anwendungen statt. Als Ergebnis wurden – inspiriert von den Infiniband-Verbs für die Kommunikation – die IOVerbs entwickelt. Die IOVerbs können I/O-Semantiken für spezifische I/O-Nutzungsszenarien ausdrücken, so dass sie von verschiedenen Speichersystemen hocheffizient umsetzbar sind. Beispiele sind Konsistenzanforderungen für sequentielles I/O und paralleles I/O sowie deren Abbildung auf Block- oder Byte-adressierbare Speicher. Analog zu POSIX werden beliebige (parallele) Anwendungen in die Lage versetzen, mit den IOVerbs Speichersysteme anzusprechen. Um die Akzeptanz zu verbessern, wurde die E/A existierender Anwendungen mit einer Kompatibilitätsschicht ebenfalls unterstützt. Dafür wurde die POSIX-Schnittstelle transparent auf IOVerbs abgebildet und entsprechende Konsistenzanforderungen mit den IOVerbs beschrieben. Ein direktes Interface um mit Byte-adressierbaren Speicherobjekten zu arbeiten, ist ebenfalls Teil der IOVerbs.

2) Basierend auf dem IML System (Infinite Memory Layer) des Fraunhofer ITWM wurde ein Ökosystem für die hocheffiziente und speicherzentrische Ein-/Ausgabe für Anwendungsszenarien in Kampagnen[11] entwickelt. IML wird seit 2011 in EU Projekten entwickelt[12] und wurde in diesem Projekt eingesetzt und um I/O Schnittstellen, Metadaten und Persistenz weiterentwickelt. Die IML Schicht präsentiert in ihrem Kern der Anwendung einen abstrakten linearen Speicher und ist in der Lage, das Speichermedium (DRAM, NVRAM) zu verbergen. Diese Ansicht ermöglicht Anwendungen die gemeinsame Nutzung von Daten auf konsistente, zuverlässige und schnelle Weise.

Als Testszenarien sollten die synthetische Benchmarks der IO500 und der I/O intensive parallele Strömungslöser CODA dienen. Im Rahmen des Projekts wurde die Skalierbarkeit der E/A von CODA signifikant verbessert, um eine neue Problemklasse skalierbar ausführen zu können.

Die produktive Inbetriebnahme des Systems auf den Rechnersystemen der beteiligten Rechenzentren, die Portierung von Anwendungen, die Skalierbarkeitsverbesserung von CODA und die Leistungsverbesserungen sollten ursprünglich messbaren Ergebnisse des Projektes sein.

Durch die verspätete Einstellung von Personal für das MCSE-Projekt und die benötigte Einarbeitung ergab sich ein allgemeiner Projektrückstand. Aufgrund dessen haben wir eine Verschiebung des Hauptmeilensteins um 6 Monate beantragt und bewilligt bekommen. Alle Projektpartner waren bestrebt, den Rückstand durch fokussierte Anstrengungen und enge Zusammenarbeit aufzuholen. Dies ist vollkommen gelungen und ergab einen erfolgreichen Abschluss des Hauptmeilensteins im September 2024. Zur weiteren Verbesserung der erzielten Ergebnisse wurde das Projekt nach dem ursprünglichen Projektende um weitere 3 Monate kostenneutral verlängert.

[11] Komplexere Workflows von der initialen Eingabe bis zur Auswertung. Diese können Wochen bis Monate andauern.

[12] `http://www.intertwine-project.eu/sites/default/files/images/INTERTWinE_D4.2.pdf`

1.4 Wissenschaftlicher und technischer Stand, an den angeknüpft wurde

Für die einzelnen Komponenten des Projekts MCSE gliedert sich der Stand der Technik in 1) E/A-Schnittstellen; 2) in-memory Speichersysteme; und 3) on-demand Speicher und Workflows.

E/A-Schnittstellen. Die POSIX-Schnittstelle ist ein bekannter Engpass bei E/A auf verteiltem Netzwerkspeicher, da sie keine Notation für Parallelität besitzt, jedoch sehr strikte Konsistenzanforderungen diktiert[13]. Es konnte bereits gezeigt werden, dass viele parallele Anwendungen so entworfen sind, sodass sie diese Art von Konsistenzanforderungen nicht benötigen. Bereits 2006 hat sich eine Working Group[14] mit der Erweiterung des POSIX Standards für parallelen I/O beschäftigt[15]. Erweiterungen der POSIX-Schnittstelle für parallele I/O wurden entworfen sind jedoch nie in den offiziellen Standard übernommen worden. Kleinere Modifikationen wie ein leichtgewichtigeres stat() ohne Rückgabe der Dateigrößen wurden umgesetzt.

Die Versuche eine High-Level API als Standard zu etablieren schlugen fehl - bspw. MPI-IO[16] oder die, von der GAUG mit unterstützte, aber nicht koordinierte, Exascale10[17]-Initiative. Seit dem Jahr 2000 findet eine Explosion von Middlewarelösungen und High-Level APIs statt und jedes Jahr werden auf Konferenzen neue Alternativen wie ADIOS [18] vorgestellt. Im Gegensatz zu diesen Ansätzen wird in diesem Projekt versucht eine Low-Level API zu erzeugen, welche effektiv High-Level APIs und beliebige parallele Use-Cases abbilden kann. Dem Konsortium ist kein Ansatz bekannt I/O analog zu IB-Verbs[19] auf Low-Level Ebene zu verallgemeinern.

In-memory Speichersysteme, wie PMEM-IO[20], ermöglichen den direkten Zugriff von CPUs auf persistenten Byte-adressierbaren Speicher. Der verteilte Speicher der Server in einem Cluster kann über verschiedene Lösungen als ein gemeinsamer Speicher abgebildet werden. Diese Abbildung kann in zwei Varianten umgesetzt werden: implizit, d.h. ohne die Kenntnis der Anwendung wird die Speicherallokation in entfernten Speicher abgebildet, oder explizit, d.h. der Programmierer muss über eine Bibliothek entfernten Speicher anfordern. KOVE[21] bietet mit KDSA eine Schnittstelle für einen globalen und (praktisch effektiv) persistenten Arbeitsspeicher an. Die Server von KOVE teilen hierbei den logischen Arbeitsspeicher in Chunks zwischen den einzelnen Servern auf. MemVerge[22] bietet über Intel Optane[23] einen beliebig großen und schnellen Speicher an. DAOS[24] bietet für Optane eine hochperformante dedizierte Speicherlösung mit Anbindungen an verschiedene High-Level APIs wie HDF5[25] an.

Die verfügbaren Lösungen sind spezialisiert auf bestimmte Speichertypen, z.B., Optane, und auf einzelne Hersteller begrenzt. Mit den IOVerbs soll eine herstellerunabhängige API geschaffen werden, welche die Gefahr des Vendor-Lock-in abwendet und eine hohe Flexibilität bietet. Mit dem MCS2 wird eine offene Lösung realisiert, welche für Kampagnen gekoppelt ist und alternative Speicherzugriffsmöglichkeiten bietet.

On-Demand Dateisyteme wie GekkoFS[26] und BeeOND[27] bieten einer Anwendung für die Laufzeit den lokalen Speicher an. Typischerweise wird hierbei für den Lauf eines Jobs (im Sinne eines Job Schedulers) das entsprechende Dateisystem hochgefahren. Solche Scratch-Systeme bieten eine skalierbare Leistung während eines Jobs. Dieses Konzept soll im Projekt auf Kampagnen erweitert und anschließend aufgezeigt werden, dass es effektiv ist, ein Speichersystem für längere Abläufe zu erzeugen und den Workflow der Kampagnen automatisiert zu managen. Im Unterschied zu Workflow-Engines wie Galaxy[28], Bpipe[29], Taver-

[13] https://www.nextplatform.com/2017/09/11/whats-bad-posix-io/
[14] https://www.pdl.cmu.edu/posix/
[15] https://www.pdl.cmu.edu/posix/docs/POSIX-extensions-goals.pdf
[16] MPI-IO: https://doku.lrz.de/display/PUBLIC/MPI-IO
[17] Exascale10: https://www.deep-projects.eu/applications/28-use-cases/239-increasing-i-o-performance-with-exascale10.html
[18] ADIOS: https://www.olcf.ornl.gov/center-projects/adios/
[19] IB-Verbs: https://docs.kernel.org/infiniband/user_verbs.html
[20] PMEM-IO: https://pmem.io/
[21] KOVE: https://kove.net/
[22] MemVerge: https://memverge.com/
[23] Intel Optane: https://www.intel.com/content/www/us/en/architecture-and-technology/optane-technology/optane-for-data-centers.html
[24] DAOS: https://docs.daos.io/v2.0/
[25] HDF5: https://www.hdfgroup.org/solutions/hdf5/
[26] GekkoFS: https://storage.bsc.es/gitlab/hpc/gekkofs
[27] BeeOND: https://www.beegfs.io/wiki/BeeOND
[28] Galaxy: https://galaxyproject.org/learn/advanced-workflow/
[29] Bpipe: http://docs.bpipe.org/

na[30] und Pegasus[31] wird im Projekt mit HPCSerA ein leichtgewichtiges Modell genutzt und explizit für die Datenverwaltung zwischen Rechenjobs angepasst, dass diese für beliebige Job-Scheduler nutzbar sind.

1.5 Zusammenarbeit mit anderen Stellen

Während des MCSE-Projektes wurde mit zahlreichen Stellen zusammengearbeitet. Zu nennen sind hier die direkten Projektpartner, die Technische Universität Dresden, das Fraunhofer ITWM, das Deutsche Zentrum für Luft- und Raumfahrt, und als assoziierter Partner die ThinkParQ GmbH. Darüberhinaus wurde im Rahmen von der HPC-Statustagung, der ISC-Konferenz und den von der GAUG organisierten Monthly Storage Talks ein fachlicher Austausch mit der interessierten breiten Fachöffentlichkeit gepflegt. Im Rahmen von Seminaren wurden außerdem Teile des Projekts beim HLRS in Stuttgart und SURF in Amsterdam in den Niederlanden präsentiert.

[30] Taverna: `https://incubator.apache.org/projects/taverna.html`
[31] Pegasus: `https://pegasus.isi.edu/`

2. Eingehende Darstellung

2.1 Verwendung der Zuwendung und erzieltes Ergebnis im Einzelnen, mit Gegenüberstellung der vorgegebenen Ziele

2.1.1 AP1: IOVerbs (Leitung: GAUG)

AP1 hatte das Ziel, die Basis für die anderen Arbeitspakete durch die Definition der IOVerbs zu liefern. Dazu wurden zunächst bestehende IO-Semantiken anhand von Use-Cases analysiert und eine neue Taxonomie aufgebaut. Basierend darauf wurden im Arbeitspaket, die IOVerbs-API und eine High-Level-API für die Anwendung in Strömungssimulationen definiert.

AP1.a Analyse & Use-Case Ausarbeitung (Leitung: GAUG/TUD) Im Arbeitspaket AP1.a standen die Analyse bestehender I/O-Semantiken sowie die systematische Erhebung repräsentativer Use-Cases für moderne HPC- und datengetriebene Workloads im Fokus. I/O-Stacks sind mehrschichtig aufgebaut und umfassen Anwendung, Middleware/High-Level-Bibliotheken, MPI-IO, POSIX/Kernel, Dateisysteme und Geräte. Bei Wechseln zwischen den Schichten gehen semantische Informationen teilweise verloren. Dadurch wird die Anwendungssicht schrittweise auf syntaktische Operationen reduziert, sodass untere Schichten die ursprüngliche Semantik nicht mehr rekonstruieren können. In der Folge müssen untere Schichten konservative Worst-Case-Annahmen treffen, um Korrektheit (insbesondere Konsistenz und Persistenz) auf den Speichersystemen zu garantieren, was sich häufig unmittelbar in Performanz-Overhead oder unnötigen Synchronisationen niederschlägt.

Im Projektverlauf wurde eine Analyse von I/O-Semantiken durchgeführt und in einem Journal-Paper veröffentlicht (Frontiers in High Performance Computing, Oeste et al. (2025)), das Terminologie und Taxonomie definiert. Die Analyse unterscheidet dabei explizit zwischen API/Feature (syntaktische Operationen) und Semantik (beobachtbares Verhalten und Interaktion von Operationen). Als zentrale semantische Kategorien wurden u.a. Parallelitäts-/Zugriffssicht (shared/exclusive), Persistenz, Konsistenz (z.B. sequentiell, session, commit, eventual) sowie zeitliche und räumliche Aspekte der Datenverwendung und Veränderbarkeit herausgearbeitet. Diese Kategorien dienten zur Ableitung der Anforderungen an eine semantikbewusste Schnittstelle.

Parallel dazu wurde eine Sammlung an Use-Cases als minimale I/O-Kernel aufgebaut und dokumentiert (MS1.1). Die Use-Cases decken verbreitete HPC-Zugriffsmuster sowie Exascale-relevante Szenarien ab. Sie sind als minimaler I/O-Kernel formuliert und dienen als Referenz für Diskussion und Implementierung. Dazu zählen u.a. Muster aus IO500 (IOR-ähnliche sequenzielle Schreib-/Lesephasen sowie Metadatenlast in mdtest), strided shared-file Zugriffe (POSIX und MPI-IO kollektiv), Streaming-Workloads (z.B. stenzil-/timesteps-basierte Ausgaben mit Double Buffering), asynchrones Checkpointing zur Überlappung von I/O und Rechenphasen sowie crash-konsistente Updates, die in POSIX typischerweise über temporäre Dateien und atomare Umbennungen realisiert werden. Ergänzend wurden datengetriebene Workloads adressiert, beispielsweise zufälliges Lesen vieler Trainingsdateien aus Verzeichnissen (readdir/random-read), um Metadaten- und Directory-Operationen für ML/AI-Szenarien abzubilden. Weiterhin wurde ein typisches Simulationsmuster identifiziert, bei dem Prozesse große, zusammenhängende Daten-Chunks kollektiv und offset-basiert in eine gemeinsame Datei schreiben. Dieses „Chunk-per-Process“-Muster im Single-Shared-File-Ansatz repräsentiert charakteristische I/O-Muster strukturierter Simulationsdaten, wie sie in CFD-Anwendungen auftreten.

Aus der Kombination von Semantik-Analyse und Use-Case-Erhebung wurden konkrete Anforderungen an die IOVerbs-Spezifikation abgeleitet, die unmittelbar in AP1.b eingeflossen sind: (i) die Fähigkeit, Semantik explizit auszudrücken (statt implizit über API-Nebenwirkungen), (ii) eine konsequent asynchrone Ausführung, um Überlappung zu ermöglichen sowie (iii) transaktionale bzw. batch-basierte Operationen, um atomare Gruppenbildung, definiertes Fehlerverhalten und Ordnungseigenschaften kontrollierbar zu machen. Ebenso zeigte sich, dass Metadaten und Gruppierungskonzepte (z.B. Verzeichnis-ähnliche Strukturen) als Schlüssel-Wert-Informationen in vielen Szenarien zentral sind, um Objekte effizient zu adressieren und zu suchen.

AP1.a liefert die fachlichen Grundlagen und Validierungsartefakte für die Entwicklung der libIOVerbs-API entlang realer Zugriffsmuster:

- Analyse und Terminologiearbeit zu I/O-Semantiken inkl. Taxonomie und Ableitung zentraler semantischer Kategorien,
- kuratierte und dokumentierte Use-Case-Sammlung als minimaler I/O-Kernel-Katalog (u.a. IO500-, Streaming-, Checkpoint- und Metadaten-lastige Muster),
- konkrete, aus Use-Cases abgeleitete Anforderungen an Semantik-Controls, Konsistenz-/Persistenzmodelle sowie Batch-/Transaktionsmechanismen als Input für AP1.b.

AP1.b API Spezifikation (Leitung: GAUG/TUD) Im Arbeitspaket AP1.b wurde die initiale Spezifikation der libIOVerbs-API als verbindliche Grundlage für die nachfolgenden Implementierungsarbeiten in AP2 erarbeitet (MS1.2). Aufbauend auf den in AP1.a herausgearbeiteten Semantik-Kategorien und den kuratierten Use-Cases war das Ziel, eine Schnittstelle zu definieren, die semantische Anforderungen explizit formulierbar macht und gleichzeitig effizient genug bleibt, um als systemnahe Basis für unterschiedliche Backends (z. B. IML, BeeGFS User-Space) zu dienen. Ein Leitprinzip der Spezifikation ist daher die Trennung zwischen (i) einem kleinen Satz von Kernobjekten/Operationen und (ii) expliziten Controls, die das beobachtbare Verhalten (Konsistenz, Ordnung, Persistenz) aus Anwendungssicht deklarieren.

Die Spezifikation definiert ein objektorientiertes Modell aus *Workspace* und *Collection* als zentrale Abstraktionen. Workspaces dienen als Namespaces und Allokations- bzw. Verwaltungsdomänen; Collections repräsentieren die eigentlichen Datenobjekte und können über Metadaten (Key-Value Paare) adressiert, gesucht und verwaltet werden. Durch diese Modellierung lassen sich sowohl dateiähnliche Muster (benannte Objekte, Verzeichnis-ähnliche Gruppierung über Metadaten) als auch objektbasierte Zugriffsmuster einheitlich ausdrücken, ohne Implementierungsdetails eines konkreten Speichersystems vorwegzunehmen.

Ein Kernbeitrag von AP1.b ist die Festlegung des Semantikmodells über *Controls*. In der C-API werden diese als Flag-basierte Steuerung umgesetzt, die u. a. Konsistenzniveaus (sequential/session/commit/eventual), Temporalität (once/multiple/bursty), Mutabilität (immutable/overwrite/append), Fehlerverhalten (continue/stop/rollback) sowie Parallelitäts- bzw. Zugriffssicht (shared/exclusive) beschreibbar macht. Damit lassen sich die aus Use-Cases bekannten Anforderungen an Überlappung, atomare Gruppenbildung und definierte Sichtbarkeit direkt abbilden. Gleichzeitig dient die Spezifikation als gemeinsamer Referenzrahmen für die Diskussion und Verifikation von Backend-Verhalten über Partnergrenzen hinweg.

Zur effizienten Umsetzung der Semantiken wurde zudem das Ausführungsmodell präzisiert: Daten- und Metadaten-Operationen sind konsequent asynchron gedacht und können in Batches gebündelt werden. Ein Batch sammelt mehrere Operationen, die dann asynchron gestartet und über Requests/Statusobjekte abgefragt werden können (wait/test/cancel). Diese Konzeption erlaubt (i) eine klare Trennung zwischen Beschreibung/Enqueue und Ausführung, (ii) eine definierte Behandlung von Fehlerfällen für Gruppen von Operationen sowie (iii) eine natürliche Abbildung auf moderne Backends, die selbst asynchron arbeiten oder mehrere Schritte zusammenfassen.

Als Ergebnis der Spezifikationsarbeiten wurde eine zweigleisige Interface-Strategie festgelegt: Neben einer modernen C++-API (zur komfortablen Nutzung in neuen Komponenten) wird eine C-API mit stabiler ABI als niedrigschwelliger Integrationspfad bereitgestellt. Beide Varianten sind dokumentiert und werden über eine gemeinsame Doxygen/Sphinx-Dokumentationskette in eine konsistente Referenz überführt. Die Dokumentation wurde im Projektverlauf sukzessive aufgebaut (Sphinx-Start: Mai 2024), wobei einzelne Teile (z. B. Collection-Abschnitte der C-API) im späteren Projektverlauf vervollständigt wurden. Parallel dazu wurde Feedback aus frühen Portierungsarbeiten (u. a. IO500-Mockup und Metadatenannahmen) als praktisches Korrektiv genutzt; insbesondere zeigte sich hierbei die Notwendigkeit, Metadaten-Konventionen und verpflichtende Attribute in der Spezifikation weiter zu präzisieren.

Zusammenfassend schafft AP1.b damit die stabile, implementierbare Grundlage für die weiteren Arbeitspakete und stellt sicher, dass die in AP1.a identifizierten Semantikbedarfe in einer prüfbaren Form vorliegen:

- initiale libIOVerbs-API Spezifikation (MS1.2) als Startpunkt für Implementierungen in AP2,
- definierte Kernobjekte (Workspace/Collection) mit Metadaten-basierter Adressierung/Lookup/Search,
- festgelegtes Semantikmodell über Controls (Konsistenz-, Zeit-, Mutabilitäts-, Fehler- und Parallelitätssemantik),

- asynchrones Ausführungs- und Batchmodell inkl. Requests/Status zur Skalierung und Überlappung,
- Dokumentationsbasis (Doxygen/Sphinx) für C- und C++-API als gemeinsame Referenz im Konsortium.

AP1.c IML Abbildung (Leitung: ITWM) Ziel von AP1.c war es, eine Abbildung der libIOVerbs-API auf den linearen Adressraum von IML zu entwickeln. Dabei geht es sowohl um die physische Speicherung der Daten als auch um eine Struktur, die die Daten selbst und deren Metadaten organisiert.

Anhand der erarbeiteten Use-Cases wurden die Semantiken für IOVerbs initial definiert. Diese wurden zusammen mit den Projektpartnern durchgehend weiter verfeinert, um die Anforderungen der Use-Cases bestmöglich zu erfüllen. Außerdem wurden weitergehende Anforderungen an die Semantik identifiziert und implementiert.

Die angestrebte Abbildung erfolgt dabei als Backend-Schnittstelle. Die IOVerbs-Backend-Schnittstelle wurde in enger Zusammenarbeit gemeinsam mit den Projektpartnern entwickelt. Zentraler Entwurfsgegenstand war die Abbildung der IOVerbs Konzepte auf den linearen Adressraum von IML. IOVerbs Collections werden dabei auf IML Segmente innerhalb von Storages abgebildet, und der Zugriff auf Teilbereiche einer Collection erfolgt mittels `iml::core::memory::Range`s.

Die Anbindung eines Backends wird mittels einer Shared Object Datei umgesetzt, welche zur Laufzeit in die IOVerbs Applikation eingebunden wird. Dieses stellt die Funktionen zur Verfügung, um das jeweilige Backend zu initialisieren und herunterzufahren. Es können Collections geöffnet, geschlossen, bearbeitet, d.h. gelesen und geschrieben sowie gelöscht werden. Außerdem können Batches dieser Operationen an das Backend übergeben werden.

Die Schnittstelle ist so konzipiert, dass mehrere Backends – auch unterschiedlichen Typs und mit verschiedenen Konfigurationen – gleichzeitig angebunden werden können. Damit wird es Applikationen ermöglicht, Daten je nach Nutzungssemantik auf verschiedene Backends aufzuteilen, z.B. gleichzeitig auf ein IML- und ein BeeGFS-Backend. Aktuell können zu einem Storage Provider drei verschiedene Storage-Backends hinzugefügt werden. Dies sind im Einzelnen, ein Storage-Backend auf dem `Heap`, im Shared Memory `SHMEM` und in einem darunterliegenden Dateisystem mittels `File`-Schnittstelle. Der Zugriff auf den Storage Provider erfolgt dabei entweder lokal über das Streaming-Protokoll oder von anderen Computern mittels TCP.

Die IML-Abbildung ermöglicht den Zugriff auf verschiedene Speichertechnologien:

- Nutzung von IOVerbs zum Zugriff auf IML.
- Auf das Storage-Backend kann sowohl lokal als auch verteilt zugegriffen werden.
- Es werden verschiedene Speichertechnologien über eine identische Schnittstelle zur Verfügung gestellt.

AP1.d High-Level API für Strömungsmechanik (Leitung: DLR) Im Rahmen des Projekts wurde eine High-Level-API entwickelt, die das Lesen und Schreiben von Strömungssimulationsdaten in Exascale-Anwendungen ermöglicht. Als Grundlage diente der bestehende I/O-Mechanismus von FSDM, dessen Schnittstellen zunächst auf ihre Eignung für die High-Level-API überprüft wurden (siehe auch den Abschnitt zu AP5.b). Diese Analyse hat ergeben, dass HDF5 (The HDF Group (2026)) als Datenformat die beste Wahl darstellt, da es weit verbreitet, für paralleles I/O optimiert und als robuste, langfristig unterstützte Lösung etabliert ist. Zudem erwies sich die Virtual File Driver (VFD)-Schnittstelle von HDF5 als vielversprechend, um das Projektziel der Anbindung von IOVerbs und IML/MCS2 an FSDM zu realisieren (siehe auch den Abschnitt zu AP5.c). Die Ergebnisse dieser Analyse wurden zudem mit Erkenntnissen aus der Fachliteratur sowie mit den I/O-Lösungen anderer Software im Bereich der Strömungssimulation abgeglichen.

Die entwickelte API basiert auf einer mehrschichtigen Architektur (siehe Abbildung 1), deren zentraler Bestandteil die im Projekt entwickelte `HDF5MeshIO`-Komponente ist. Diese hat die Aufgabe, Netze und Daten aus Strömungssimulationsanwendungen auf das HDF5-interne Datenmodell, bestehend aus Gruppen, Datensätzen und Attributen, abzubilden und damit das Layout festzulegen, in dem die Simulationsdaten in einer HDF5-Datei abgelegt werden. Auch hierbei wurde sich an dem bereits in FSDM etablierten Datenlayout orientiert. Um eine breite Anwendbarkeit der API zu gewährleisten, wurde diese Logik durch die Einführung einer geeigneten Abstraktionsschicht von den Implementationsdetails der Netzklassen spezifischer Strömungssimulationsanwendungen separiert. So können neben den im Projekt konkret betrachteten

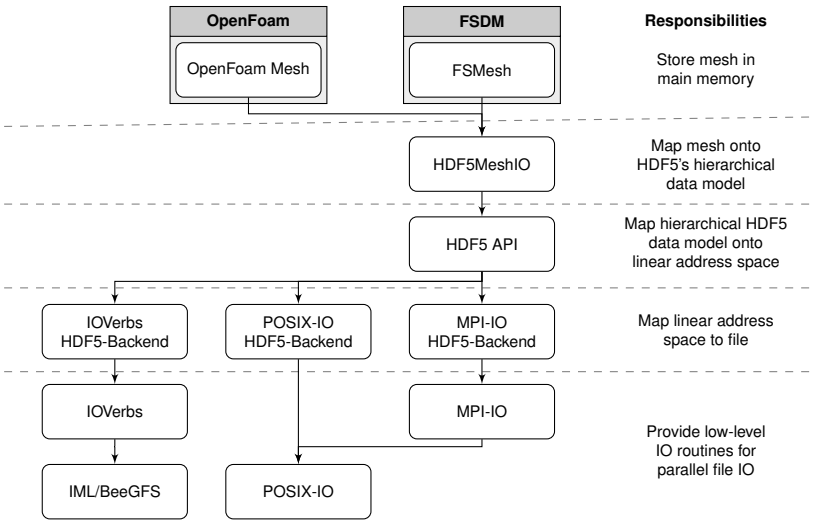

Abbildung 1: Software Architektur für High-Level-I/O-API für Strömungsmechanik

Softwareprodukten FSDM und CODA auch andere Simulationsumgebungen Gebrauch von der entworfenen API machen.

Die `HDF5MeshIO`-Komponente baut auf der HDF5-C-API auf, um die erzeugte Hierarchie aus Gruppen, Datensätzen und Attributen auf einen linearen Adressraum abzubilden, der schließlich in einer HDF5-Datei persistiert wird. Die Art und Weise, wie dieser lineare Adressraum auf eine HDF5-Datei abgebildet wird, lässt sich mittels der von der HDF5-API zur Verfügung gestellten VFD-Schnittstelle anpassen. So konnte durch die Bereitstellung geeigneter VFDs eine Anbindung von IML/MCS2 und IOVerbs an die High-Level-I/O-API umgesetzt werden (für Details siehe den Abschnitt zu AP5.c und AP5.d).

Insgesamt wurde somit eine modulare API geschaffen, die sich flexibel an diverse Einsatzzwecke anpassen lässt. Im Bereich der Strömungsmechanik können mit der zur Verfügung gestellten `HDF5MeshIO`-Komponente und einer generischen Schnittstelle zur Anbindung unterschiedlicher Netzklassen auf einfache Weise auch andere Strömungslöser auf dem bereitgestellten Software-Stack aufbauen, um Netze und Simulationsdaten zu importieren und zu exportieren. Durch die flexibel und unabhängig gehaltene Backend-Schnittstelle können diese sowohl HDF5-eigene VFDs, beispielsweise für POSIX oder MPI-IO, verwenden, um in klassische Dateisysteme zu schreiben, als auch die im Projekt entworfenen spezialisierten VFDs nutzen, um ihre Daten in IML oder BeeGFS abzulegen. Dabei können semantische Anforderungen an das I/O-Verhalten transparent mittels IOVerbs an diese Speicherbackends weitergegeben werden. Darüber hinaus können die entwickelten VFDs für IML und IOVerbs auch in anderen Domänen eingesetzt werden. Insbesondere bestehende Software, die bereits HDF5 für ihr Datei-I/O verwendet, kann mit minimalem Aufwand an die libIOVerbs-Bibliothek angebunden werden. Beim Öffnen der Datei kann diese ihre I/O-Semantiken mittels einer File Access Property List (FAPL) über den IOVerbs-VFD an das IOV-Backend sowie das verwendete Speicherbackend (z.B. IML oder BeeGFS) weitergeben. Dies demonstriert, dass eine Anbindung bestehender Software – unabhängig von der Domäne – an IOVerbs mit geringem Aufwand möglich ist.

2.1.2 AP2: Implementierung (Leitung: TUD)

AP2.a Implementation libIOVerbs (Leitung: TUD) Im Arbeitspaket AP2.a wurde die Referenzimplementierung der IOVerbs Bibliothek (libIOVerbs) als technische Grundlage für die im Projekt definierte libIOVerbs-API erarbeitet. Ziel war es, eine schlanke, systemnahe Bibliothek bereitzustellen, die I/O-Operationen nicht nur ausführt, sondern zusätzlich explizite semantische Informationen (z. B. Konsistenz-, Persistenz-, Mutabilitäts- und Parallelitätseigenschaften) bis zu nachgelagerten Speicher- und Dateisystemschichten transportiert. Damit adressiert libIOVerbs direkt die Projektherausforderung, Anwendungssicht und Systemumsetzung über einen wohldefinierten Übergang zu koppeln, ohne die Flexibilität für unterschiedliche Backends einzuschränken.

Die Implementierung folgt den im Rahmen von AP1 spezifizierten Kerndesignprinzipien: (i) eine konsequent asynchrone Programmierschnittstelle, (ii) transaktionale Batch-Ausführung zur Gruppierung mehrerer

Operationen inkl. definierter Fehlersicht (z. B. Continue/Stop/Rollback) sowie (iii) die ausdrückliche Modellierung von Semantik über *Semantic Controls*. Inhaltlich wurde das Datenmodell in Workspaces (isolierte Namespaces/Allokationsdomänen), Collections (Container/Objekte innerhalb eines Workspaces) und Chunks (Bytebereiche als Datenabstraktion) gegliedert. Dadurch lassen sich sowohl klassische Dateisystemmuster als auch objekt- oder speicherorientierte Zugriffe abbilden, während die eigentliche Umsetzung dem jeweils gewählten Backend überlassen bleibt.

Ein wesentlicher Teil von AP2.a war die Realisierung einer eigenständigen Metadatenkomponente in libIOVerbs. Metadaten werden als Key-Value-Paare geführt und dienen sowohl der Adressierung (Workspace-/Collection-Name, UUIDs) als auch der Suche und der Ablage wissenschaftlicher Attribute. Die Architektur sieht dabei ein verteiltes Metadaten-Backend (aktueller Referenzpfad: ECTD) sowie optionales lokales Caching (RocksDB oder deaktivierbar) vor. Diese Trennung ermöglicht Konsistenz- und Performanzanforderungen kontrolliert zu variieren und gleichzeitig die Bibliothek in unterschiedlichen Umgebungen einsetzbar zu halten.

Für die praktische Nutzbarkeit wurden die Bibliothek und das Build-/Packaging-Setup so umgesetzt, dass sie als eigenständige Komponente integrierbar ist: Die CMake-basierte Buildkette erlaubt das gezielte Aktivieren von Dokumentation, Tests und Beispielen sowie die Auswahl der Metadatenbackends. Neben einer modernen C++-API wird eine C-API bereitgestellt, um eine niedrigschwellige Anbindung für existierende HPC-Softwarestapel zu ermöglichen. Dabei wurde die C++ API wurde auf eine idiomatische C-API abgebildet, wobei alle Fähigkeiten der C++ API, asynchrone Batch-Ausführung, Semantiken und Fehlerkontrolle erhalten bleiben. Die C-API hat eine eigene Test-Suite und grundlegende Metadaten Operationen zu gewährleisten. Zur frühen Validierung und Entkopplung von externen Speichersystemen wurde zudem ein Dummy-Backend als No-Op-Referenz implementiert, das Tests und erste Integrationsarbeiten (z. B. für Prototypen und CI) erlaubt. Als zentrale Ergebnisse von AP2.a wurden damit insbesondere bereitgestellt:

- eine libIOVerbs Kernbibliothek mit Workspace/Collection/Chunk-Abstraktionen und transaktionaler Batch-API,
- eine implementierte Steuerungsschicht für I/O-Semantiken (*Semantic Controls*) zur expliziten Beschreibung von Konsistenz, Fehler- und Parallelitätsverhalten,
- eine Metadatenimplementierung auf Basis von Key-Value-Strukturen inkl. Namens-/UUID-Konzept und Such-/Lookup-Operationen,
- eine konfigurierbare Metadatenpersistenz mit verteiltem Backend (ETCD) und optionalem lokalem Cache (RocksDB),
- ein reproduzierbares Build-/Test-Setup (CMake-Optionen, Unit-Tests/Beispiele) sowie ein Dummy-Backend zur frühzeitigen Validierung.

Zusammenfassend liefert AP2.a damit die funktionale Basis für die weiteren Umsetzungen in AP2 (Backends, Pre-loading, FUSE) sowie für die Anwendungsportierungen und Benchmark-Integrationen: Die zentralen Konzepte (asynchron, Batch/Transaktionen, *Semantic Controls*, Metadatenverwaltung) sind in einer konsistenten Bibliotheksstruktur umgesetzt und über Build-Optionen reproduzierbar nutzbar. Offene Punkte betreffen insbesondere die sukzessive Härtung der Schnittstellen, die Feinspezifikation verpflichtender Metadatenattribute sowie die fortlaufende Anbindung und Optimierung realer Daten-Backends.

AP2.b IML Backend (Leitung: ITWM) Um die Nutzung von IOVerbs mit IML für Anwendungen nutzbar zu machen, wurde im AP2.b eine User-Space Library implementiert. Aufbauend auf dem IML-Kern wurde ein Backend für IOVerbs implementiert, das als User-Space Library bereitgestellt wird. Die von einem Backend zu implementierenden Funktionen werden von IOVerbs deklariert und von den jeweiligen Backends implementiert.

Das Design des Metadaten-Managers folgt drei zentralen Leitprinzipien: (i) strikte Trennung von Metadaten- und Datentransport, (ii) asynchrone Ausführung aller nicht-lokalen Operationen und (iii) horizontale Skalierbarkeit durch Datenverteilung und Multi-Instanz-Betrieb.

Der Metadaten-Manager koordiniert ausschließlich Zuordnungen und Zugriffe; er ist niemals selbst am Nutzdatentransport beteiligt – Daten werden stets direkt zwischen den beteiligten Speicher-Instanzen

ausgetauscht. Metadaten- und Datenkanal sind daher vollständig unabhängig voneinander und unterstützen jeweils parallelen Zugriff. Die Anzahl der Threads pro Endpunkt ist konfigurierbar, sodass das Backend an unterschiedliche Systemarchitekturen und Nutzungsmuster angepasst werden kann. Alle nichtlokalen Befehle sind asynchron spezifiziert, um die Überlappung von Datentransport und Berechnung zu ermöglichen. Zur optimalen Ausnutzung der Netzwerkbandbreite werden Daten über mehrere angebundene Speicher-Instanzen verteilt. Der Algorithmus zur Verteilung der Daten ist ebenfalls konfigurierbar. Das Design sieht zudem die gleichzeitige Anbindung mehrerer Provider-Instanzen vor, um Engpässe auf einem einzelnen Metadaten-Knoten zu vermeiden. Für effizienten Datentransfer ist ein Zero-Copy-Pfad über `backend_allocate()` im Design enthalten: Puffer in IML-verwaltetem Speicher werden ohne Kopiervorgang direkt übergeben. Für Daten außerhalb des IML-Adressraums wird ein konfigurierbarer Buffer-Pool genutzt, dessen Verwendung für die Anwendungsschicht transparent bleibt.

Das zentrale Design des IML-Backends ist eine Client-Server Architektur. Der in AP3.a implementierte Metadaten-Manager (`iml::iov_backend::Provider`) verwaltet die angebundenen Storages und koordiniert den Zugriff auf Collections. Die angebundenen Storages können beliebigen Typs sein – es genügt, die Schnittstelle `iml::core::storage::is_implementation` zu implementieren. Die unterstützten Storage-Typen sowie zusätzliche Erweiterungen (`Virtual`, `Import_C_API`, `Trace`) sind in AP3.c beschrieben.

Die an das `iov_backend` angebundenen Storage-Server können jeweils unterschiedlichen Typs und beliebig groß sein und mit unterschiedlichen Persistenz-Charakteristiken konfiguriert werden (siehe AP3.b). Lokalität und Kommunikationsprotokoll sind ebenfalls für jeden Storage individuell wählbar.

Die Applikation kann beliebig viele `iml::iov_backend::Client`s instanziieren, um auf den hinzugefügten Storages zu arbeiten.

Für die Integration mit der libIOVerbs-Bibliothek wurde zusätzlich eine Metadaten-Datenbank (`IOV_Database`) entwickelt. Diese verwaltet die Zuordnung von IOVerbs Collection-UUIDs zu den internen IML Collection-IDs und ermöglicht die Organisation der Daten in Workspaces. Kommandozeilen-Werkzeuge zum Erstellen, Befüllen und Abfragen der Datenbank stehen zur Verfügung. Diese sind im Einzelnen:

- `iml_iov_backend_iov_add_entry_to_database` fügt einer Datenbank einen Eintrag hinzu.
- `iml_iov_backend_iov_print_database` gibt den Inhalt einer Datenbank aus.
- `iml_iov_backend_iov_make_empty_database` legt eine leere Datenbank an.
- `iml_iov_backend_iov_remove_entry_from_database` entfernt einen Eintrag aus einer Datenbank.

Damit ist die Nutzung von IML in Anwendungen und ebenso die manuelle Bearbeitung der Datenbank, wie im AP geplant, möglich.

AP2.c BeeGFS User-Space Backend (Leitung: GAUG/U) Ziel von AP2.c war die Integration des in AP3.d entwickelten BeeGFS User-Space-Clients als vollwertiges Storage-Backend für libIOVerbs. Hierzu wurde untersucht und umgesetzt, wie die im Rahmen von AP1 spezifizierte IO-Verbs-API im BeeGFS-Client unterstützt und effizient auf BeeGFS-spezifische RPC-Operationen abgebildet werden kann.

Die Architektur der Basisversion des BeeGFS User-Space Client wurde überarbeitet und grundlegend modularisiert. Der bestehende Code wurde in ein asynchrones BeeGFS Backend (`beegfs_backend`) in Rust und ein Frontend zum Abfangen von Systemaufrufen (`beegfs_intercept`) aufgeteilt. Diese Trennung erlaubt eine klare Entkopplung der Storage-Logik von der API-Anbindung und bildet die Grundlage für eine flexible Backend-Integration.

Auf dieser Basis wurde ein neues libIOVerbs Backend (`beegfs_iov`) entwickelt, das die von libIOVerbs spezifizierten Operationen auf BeeGFS spezifische RPCs abbildet. Damit kann ein BeeGFS Speichersystem asynchron, ohne Kernel-Komponente und direkt aus libIOVerbs-basierten Applikationen genutzt werden.

Abbildung 2 zeigt die beiden unterstützten APIs, über die User-Space Applikationen den BeeGFS Client direkt nutzen können.

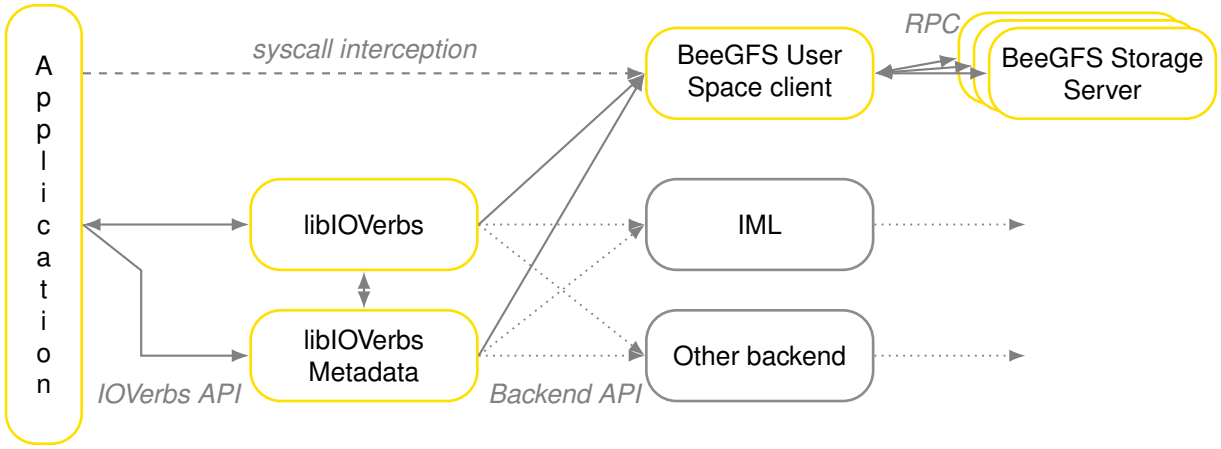

Abbildung 2: BeeGFS Komponenten im Gesamtkontext des I/O Pfades

Zur dynamischen Backend-Registrierung wurde die Funktion `iov_backend_ops()` implementiert, die Funktionszeiger für die vom Backend unterstützte Funktionalität registriert, insbesondere auch eine Initialisierungsroutine für das Backend mit backend-spezifischer Konfiguration. Auf diesem Weg können Applikationen, die libIOVervbs nutzen, Backends zur Laufzeit laden und konfigurieren, was die Flexibilität des Gesamtsystems und eine gleichzeitige Nutzung verschiedener Backends ermöglicht.

Die Interoperation zwischen der in C++ implementierten libIOVerbs und dem in Rust implementierten BeeGFS Backend erforderte eine Überbrückung verschiedener Sprachparadigmen, insbesondere hinsichtlich typenbasierten Polymorphismus in C++ und unterschiedlicher Asynchronitätsmodelle. Hierzu wurde nach eingehender Analyse und Evaluation eine schlanke FFI-Schicht in C++ implementiert, die eine sichere Weitergabe von State und den damit verbundenen Synchronisationsmechanismen und der von der Applikation geforderten Semantiken über Sprachgrenzen hinweg ermöglicht. Der Status asynchroner Operationen kann durch diese Schicht transparent aus dem Backend abgefragt werden.

Diese FFI-Implementierung zeigt, dass eine Anbindung von libIOVerbs-Backends nicht auf C++ beschränkt ist und dient als Blaupause für zukünftige Backend-Implementierungen in Rust oder anderen Sprachen.

Abbildung 3 stellt den Kontrollfluss zwischen `beegfs_iov` (links oben), `beegfs_intercept` (rechts oben) und dem gemeinsamen `beegfs_backend` dar. Auf `beefgs_iov` Seite liegt unter der FFI-Schicht eine Rust Implementierung der libIOVerbs Backend API, die IOVerbs spezifische Operationen generalisiert und an das geteilte, asynchrone `beegfs_backend` weitergibt. Im BeeGFS Backend werden die Daten dann mittels BeeGFS RPC auf mehreren Storage Servern gespeichert.

Für die Kommunikation zwischen BeeGFS User-Space Client und den Storage Servern konnte ein bereits für den ebenfalls in Rust implementierten `beegfs-mmgmtd` zur Verfügung stehender Connection-Pooling-Mechanismus größtenteils wiederverwendet werden. Dadurch werden Verbindungen zu den Servern nur einmal aufgebaut und dann persistent vorgehalten und bei Bedarf wieder verwendet, was den für den Verbindungsaufbau nötigen Overhead reduziert und somit die Operationslatenz verringert.

Zum Projektende sind die Grundfunktionalitäten der libIOVerbs API (`collection_open, collection_close, collection_delete, read, write`) im BeeGFS Backend implementiert und mithilfe Backend spezifischer, aber auch allgemeiner Tests getestet. Speziell für libIOVerbs modifizierte Applikationen können erfolgreich gegen BeeGFS ausgeführt und funktional sowie hinsichtlich ihrer Performanz validiert werden.

Für Zero-Copy-I/O und die damit verbundene Unterstützung zur Verwaltung Backend spezifischer Buffer mittels `allocate` und `free` liegt ein Merge Request vor, dessen abschließende Validierung im Projektverlauf nicht mehr erfolgen konnte. Eine Erweiterung um Batch-Operationen stellt einen naheliegenden nächsten Entwicklungsschritt dar und bietet Potential für weitere Performanzoptimierungen in einem möglichen Folgeprojekt.

Damit wurde das Ziel von AP2.c, die Etablierung von BeeGFS als funktionsfähigem und asynchronem libIOVerbs-Backend erfolgreich erreicht.

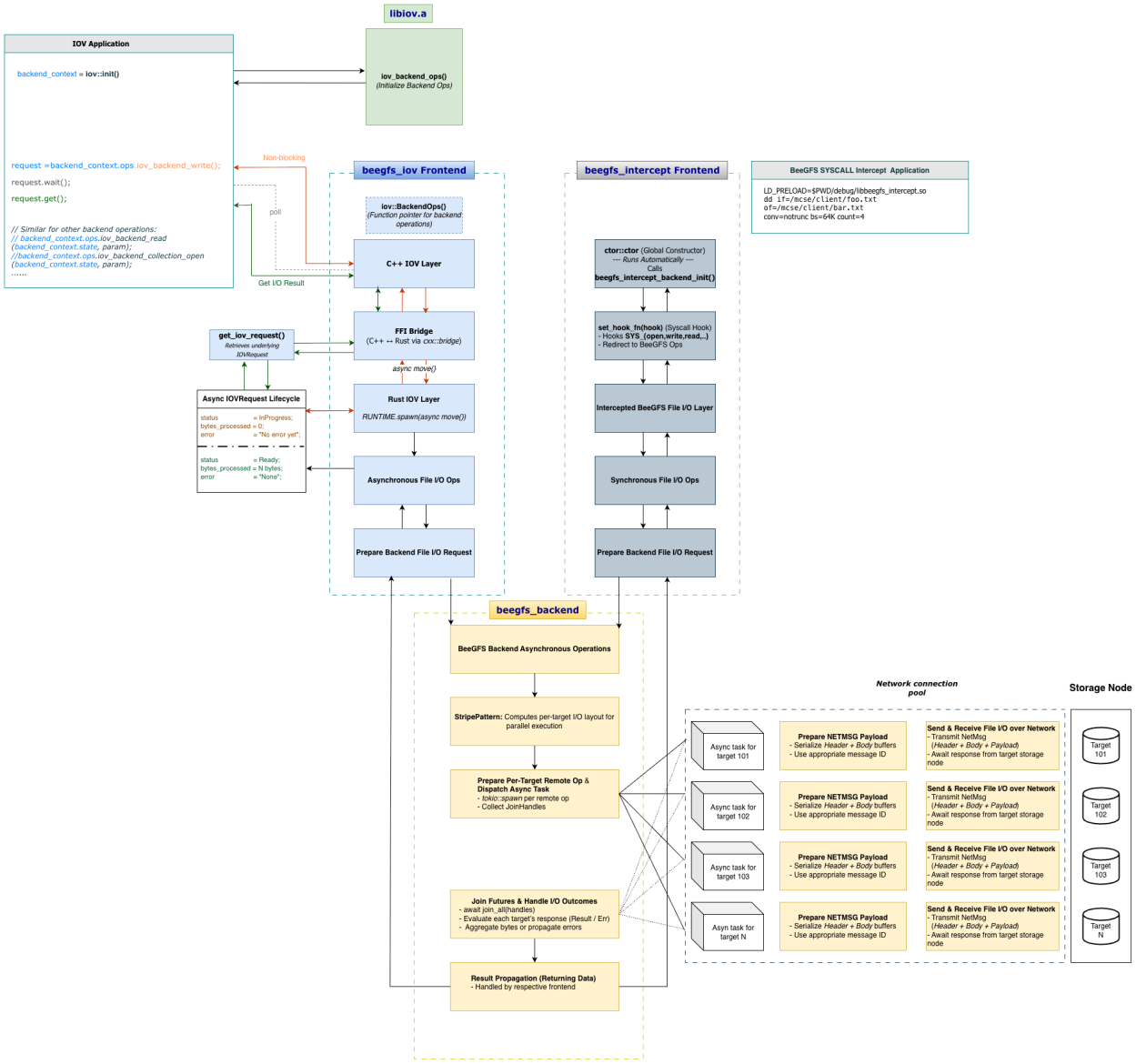

Abbildung 3: Interaktion der verschiedenen BeeGFS Frontends mit dem asynchronen Backend

AP2.d ESDM Frontend (Leitung: GAUG) Planmäßig sollte die Earth-System Data Middleware (ESDM) um ein IOVerbs-Backend erweitert werden. ESDM bietet als Plugin von NetCDF die Möglichkeit für beliebige NetCDF Anwendungen die IOVerbs zu nutzen und stellt damit für die IOVerbs ein Frontend bereit. Die dabei erarbeiteten Code Veränderungen sollten im Anschluss im ESDM Repository auf GitHub für die Community zur Verfügung gestellt werden.

Neuere Versionen von NetCDF nutzen alternativ zu ESDM auch HDF5 als Backend, wobei HDF5 in der Wissenschaft weit verbreitet ist. Es ist auch die Basis für die Integration von IOVerbs in AP5.a. Aus diesem Grund wurde auch die Entwicklung des ESDM Frontends im Laufe des Projekts niedriger priorisiert, da bereits in AP5.a eine Implementierung als Virtual Object Layer in HDF5 vorgenommen wurde. Wie bereits in AP1.d beschrieben, steht damit über HDF5 eine sogar noch weiter als ESDM verbreitete Middleware zur Integration vielfältiger Anwendungen zur Verfügung.

AP2.e POSIX Pre-loading Bibliothek (Leitung: GAUG) Zum Erfolg der IOVerbs ist die Integration von Legacy-Anwendungen unerlässlich. In AP2.e wird dabei der POSIX pre-loading Mechanismus verwendet, der einen gezielten Austausch von Funktionalität in bestehenden Applikationen ermöglicht, ohne dabei eine

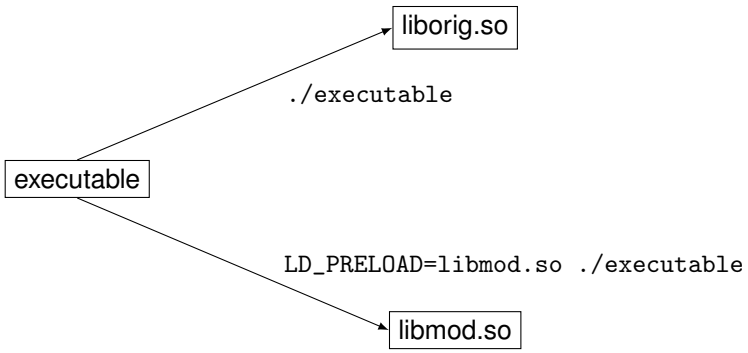

Abbildung 4: Mechanismus von Pre-loading

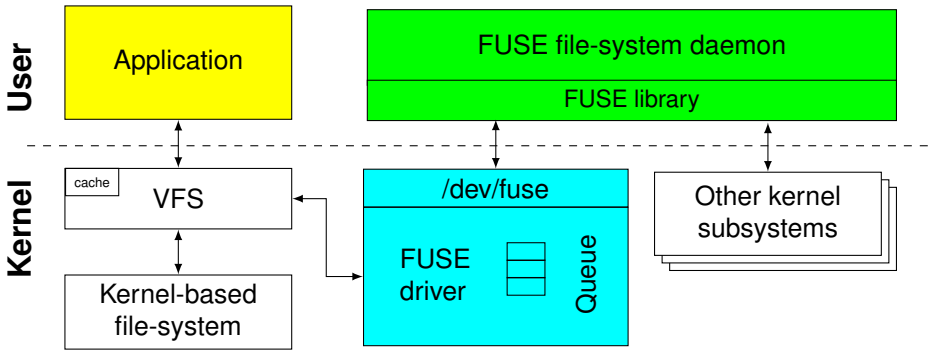

Abbildung 5: Mechanismus von FUSE (adopted from Vangoor et al. (2017))

Neukompilierung erforderlich zu machen (siehe Abbildung 4). Dadurch wird es auch möglich, bestehende Anwendungen mit IOVerbs-basierten Speichersystemen kompatibel zu machen. Die POSIX Pre-loading Bibliothek dient der niederschwelligen Integration von IOVerbs in bestehende Applikationen, die die klassischen POSIX Dateioperationen wie open, read und write, verwenden. Ohne Neukompilierung können solche Anwendungen ein MCSE auf einem konfigurierbaren Workspace, anstatt auf einer der üblichen Verzeichnisstruktur arbeiten.

Zur Demonstration der Funktionalität der C-API wurde diese in der pre-loading Bibliothek implementiert. Um die Implementation der POSIX Funktionen zu ermöglichen, wurde zunächst eine Bibliothek erstellt, iov-posix, die die POSIX-Semantiken mittels der IOVerbs implementiert. Die pre-loading-Implementation überschreibt dann die POSIX Funktionen mit den jeweiligen äquivalenten aus dieser POSIX Bibliothek. Außerdem erwies sich iov-posix als nützlich, um den FUSE-Client in AP2.f zu implementieren.

Das Arbeitspaket stellt damit die Möglichkeit zur Integration von Legacy-Applikationen bereit.

AP2.f FUSE-Client (Leitung: GAUG) Als weitere Möglichkeit der Integration von Legacy-Anwendungen wurde im Projektplan das Mounten von IOVerbs-basierten Speichersystemen in der allgemeinen Verzeichnisstruktur in AP2.f vorgesehen. Zur Vereinfachung der Implementierung wurde, wie geplant, auf die Realisierung als in Kernel-Dateisystem verzichtet und stattdessen das Mounten durch die Implementierung eines Dateisystems im User-Space (FUSE) realisiert (siehe Abbildung 5). Im Projektverlauf wurde ein FUSE-Client initial basierend auf iov-posix aus AP2.e implementiert, der es ermöglicht, ein Workspace in einer Unix Verzeichnisstruktur zu mounten. Dadurch kann beispielsweise anschaulich mit typischen Kommandozeilenprogrammen mit einem Workspace interagiert werden.

Die Metadaten Schlüssel-Wertepaare der Collections werden dabei auf Ordner- und schließlich den Dateinamen abgebildet, sodass Nutzende Collections nach Metadaten filtern können, in dem sie in die jeweiligen Ordner, bspw. (project/mcse/experiment/), absteigen. Es werden dort nur die Collections oder weitere Schlüssel, gelistet, die die jeweiligen Zuordnungen erfüllen.

Damit wurde im Projekt eine weitere Möglichkeit für Legacy-Anwendung zum Zugriff auf IOVerbs-basierte Speichersysteme geschaffen.

AP2.g Optimierung POSIX-Clients (Leitung: GAUG) In AP2.g war die Optimierung der iov-posix aus AP2.e geplant. Aufgrund von Verzögerungen im Projektverlauf konnte diese nur begonnen, allerdings nicht abgeschlossen werden.

2.1.3 AP3: Memory-Centric Storage System (Leitung: ITWM)

AP3 beinhaltet die Weiterentwicklung der bereits existierenden IML Layer zum Memory-Centric Storage System. Zusätzlich wird im Arbeitspaket der User-Space Client von BeeGFS implementiert.

AP3.a Entwicklung des IML Metadatenmanagers (Leitung: ITWM) Im AP3.a wurde der IML Metadaten-Manager implementiert, der den vollen Funktionsumfang von IOVerbs unterstützt und einen fein granulierten Zugriff auf die Daten erlaubt. Der `iml::iov_backend::Provider` erlaubt Zugriff auf Collections via `iov::Collection` und auf Ranges innerhalb dieser Collections via `iml::core::memory::Range`. Außerdem können Collections erstellt, gelöscht und dynamisch mittels einer Append-Operation vergrößert werden.

Ein zentraler Punkt für die optimale Leistungsfähigkeit ist der Zero-Copy Transport, den IML mittels `backend_allocate()` zur Verfügung stellt. IML nutzt zusätzlich einen Buffer-Pool für den Datentransport von Daten, die nicht in von IML gemanagtem Speicher liegen. Dieser Pool ist konfigurierbar, wobei diese Unterteilung keine Auswirkungen auf die Schnittstelle hat.

Des Weiteren ist der Metadaten-Manager niemals am Datentransport beteiligt, da die Daten direkt zwischen den Speicher-Systemen transportiert werden. Der Datentransport ist parallelisiert, um die Bandbreite des Mediums möglichst optimal ausnutzen zu können. Jedoch kann die Anzahl an parallelen Transportvorgängen limitiert werden, um keinen Endpunkt zu überlasten.

Gleichzeitig sollte der Datentransport nie die Steuerung des Systems behindern und vice versa. Daher besitzen sowohl die Client- als auch die Speicher-Instanzen getrennte und unabhängige Kommunikationskanäle: einen Kanal für Metadaten und einen Kanal für den Datentransport. Um die Skalierbarkeit zu verbessern, erlauben grundsätzlich alle Kommunikationskanäle parallelen Zugriff. Das trifft auch auf den Metadaten-Kanal des Metadaten-Managers zu. Alle Endpunkte dieser Kanäle sind konfigurierbar in der Anzahl ihrer Threads. Somit kann das IML-Backend an die Systemarchitektur und die erwarteten Nutzungsmuster der Anwendungen angepasst werden.

Ein weiteres relevantes Designprinzip von IML ist die Asynchronität aller nicht-lokalen Befehle. Dies ermöglicht es, dass auch nicht-lokaler Datentransport und Berechnungen voneinander unabhängig sein können.

Skalierbarkeit und Performanz werden weiter verbessert, indem die Daten auf den angebundenen Speicher-Instanzen verteilt werden. So werden Engpässe verhindert und die Bandbreite des gesamten Netzwerks kann besser ausgenutzt werden. Des Weiteren erlaubt IML die Anbindung mehrerer Metadaten-Manager-Instanzen.

Zusammenfassend wurde in AP3.a auf der Arbeit in AP2 aufbauend IML weiterentwickelt. Der notwendigen Funktionalitäten um IML als Kampagnen Speicher zu nutzen wurden entwickelt. Darüber hinaus wurde das TRL mit weiteren Funktionen, verbesserter Performanz, höherer Skalierbarkeit und verbesserter Stabilität weiter erhöht. Die dazugehörigen Tests und Dokumentation wurden ebenfalls erweitert.

AP3.b Integration von Persistenzmechanismen in das IML (Leitung: ITWM) Vor Projektbeginn konnten Daten auf flüchtigen Speichermedien von IML nicht nach einem Neustart des Speichersystems wiederhergestellt werden. In AP3.b wird deshalb um Persistenzmechanismen erweitert.

IML unterstützt sowohl flüchtige als auch nicht-flüchtige Speichermedien (siehe AP3.c für die unterstützten Storage-Typen). Um das Nutzungsspektrum weiter zu vergrößern, unterstützt IML die gleichzeitige Nutzung verschiedenartiger Speicher. Da der `Files` Storage auf beliebigen Dateisystemen arbeitet, können auch Direct-Access-fähige Dateisysteme auf NVDIMMs oder anderen persistenten Speichermedien genutzt werden.

Für nicht-flüchtige Speichermedien ermöglicht IML eine Persistierung der Daten auch nach dem Herunterfahren aller IML Prozesse. Der `Files` Storage nutzt hierfür einen mmap-basierten Zugriff. Bei einem Neustart wird das Speicherverzeichnis automatisch gescannt und alle vorhandenen Segmente werden wiederhergestellt, ohne dass die Applikation manuell Daten laden muss.

Die Persistenz-Semantik ist pro Segment individuell konfigurierbar. Jedes Segment kann entweder als persistent (`OnRemove::Keep`) oder als flüchtig (`OnRemove::Remove`) konfiguriert werden. Diese Konfiguration

kann auch zur Laufzeit geändert werden. Dadurch können Applikationen Segmente, die zunächst persistent angelegt wurden, nachträglich als flüchtig markieren, wenn die Daten nicht mehr benötigt werden.

Die Lebensdauer der Daten ist an die Lebensdauer des zugehörigen `iml::Storage` gekoppelt, analog zur RAII-Semantik in C++. Ein Anwendungsfall hierfür ist die Erzeugung temporärer Daten, die nur während eines Kampagnen-Zwischenschritts benötigt werden. Manuelles Buchhalten und Löschen nicht mehr benötigter Daten entfällt.

Zusätzlich zur Persistenz der Nutzdaten ist auch der Zustand des Metadaten-Managers serialisierbar. Dieser kann als Datei gespeichert und beim Neustart erneut geladen werden. Dies ermöglicht eine vollständige Wiederherstellung des Systems, – inklusive aller Collections, Storage-Zuordnungen und Metadaten – ohne dass die Applikationen den Zustand selbst verwalten müssen. In Kombination mit der Datenpersistenz des `Files` Storage ermöglicht dies den Einsatz von IML als komplette Speicherlösung.

Diese Persistenzmechanismen sind sowohl für das `iml::block_device` als auch für das `iml::iov_backend` implementiert und sind ein zentraler Aspekt, um effiziente Berechnungskampagnen, wie in AP4 angedacht, umzusetzen. Damit erlauben die Änderungen erst die in AP4 geplanten Kampagnen bzw. die Entwicklung von MCS2 und die Ziele von AP3.b wurden erreicht.

AP3.c Implementation des MCS2 Direct Client Interface (Leitung: ITWM) AP3.c zielt darauf ab, die bestehende rudimentäre Client-Schnittstelle mit einer speicherbasierten Semantik (malloc, free, put, get) zu verbessern und zu erweitern, um die Anforderungen von IOVerbs gezielt zu unterstützen. Darüber hinaus wird diese Client-Schnittstelle genutzt werden, um das gesamte System zu testen und nützliche Werkzeuge, z.B., eine Debugging CLI, zu entwickeln, um die Robustheit des Systems zu verbessern.

Die Client-Schnittstelle des IML-Kerns bietet Funktionen, um Speichermedien unterschiedlichen Typs und beliebiger Lokalität (`iml::Storages`) in die aktuelle Applikation einzubinden und diese somit dort zugreifbar zu machen. Die zur Zeit unterstützten Typen sind `Heap`, `SHMEM` und `Files`.

Der IML-Kern bietet ein Interface mit speicherbasierter Semantik: `segment_create` und `segment_remove` analog zu `malloc` und `free` sowie `memory_put` und `memory_get` für den Datentransport zwischen Storages. Auf lokale Segmente kann mittels `iml::Chunk`s direkt zugegriffen werden. Ein `iml::Chunk` ist semantisch vergleichbar mit einem `std::span`.

Abhängig vom Typ des Speichers und der Lokalität des zugreifenden Prozesses kann auf Daten ggf. direkt und ohne Kommunikation zugegriffen werden. Auf `Heap` Speicher kann nur innerhalb des gleichen Prozesses direkt zugegriffen werden. `SHMEM` Speicher erlaubt zusätzlich den Zugriff von anderen Prozessen, die auf der gleichen Hardware ausgeführt werden. `Files` Speicher ermöglicht einen Lokalität-agnostischen direkten Lese- und Schreibzugriff.

Neben den Standard-Storages wurden zusätzliche Storage-Typen entwickelt, um die Erweiterbarkeit und Testbarkeit des Systems zu verbessern. Ein `Virtual` Storage ermöglicht es, Storage-Implementierungen zur Laufzeit via Shared Object Files einzubinden, ohne das System neu kompilieren zu müssen. Ein `Import_C_API` Storage erlaubt die statische Anbindung reiner C-Implementierungen. Außerdem wurde ein `Trace` Storage implementiert, der beliebige andere Storage-Typen wrappen und alle Aufrufe aufzeichnen kann. Dieser ist insbesondere für das Debugging und die Performanzanalyse nützlich.

Für die Kommunikation zwischen den verteilten Komponenten wurde ein eigenes RPC-Framework entwickelt. Dieses unterstützt synchrone und asynchrone Aufrufe, parallele Multi-Client-Operationen und konfigurierbare Protokolle (TCP, Unix Domain Sockets). Darauf aufbauend implementiert eine Control-Schicht das Protokoll zur Fernsteuerung von Storages: Segmente können remote erstellt und entfernt, Kapazitäten abgefragt und Dateioperationen an Storage-Server delegiert werden. Der eigentliche Datentransport erfolgt über eine separate Transport-Schicht. Neben der Standard-Implementierung auf Basis von ASIO wurde eine LibFabric-Transport-Implementierung entwickelt, die das RDM-Modell (Reliable Datagram) nutzt und zur Laufzeit verschiedene Fabric-Provider (z.B. TCP, verbs) unterstützt. Dies ermöglicht den Einsatz von Hochgeschwindigkeitsnetzwerken wie InfiniBand.

Um das Client-Interface nutzerfreundlicher zu gestalten, wurden darauf aufbauend mehrere Präsentationsschichten entwickelt. Das `iml::block_device` stellt einen Metadaten-Server zur Verfügung. Einzelne Prozesse können lokal angelegten Speicher (`iml::Storage`) beim Metadaten-Server anmelden. Dieser Speicher ist dann von allen Applikationen zugreifbar, die mit dem Metadaten-Server verbunden sind.

Der Metadaten-Server bietet Zugriff auf den Speicher mittels `iml::block::ID`s. Somit müssen teilnehmende Applikationen Typ und Lokalität des jeweiligen Speichers nicht kennen. Stattdessen scheint der zur Verfügung stehende kontinuierliche Speicherraum für den Nutzer unendlich (IML: Infinite Memory Layer).

Applikationen müssen den Datentransfer demnach nicht selbst handhaben. Stattdessen stellt der Metadaten-Server `read` und `write` Operationen zur Verfügung. Dies ermöglicht es, den Datentransfer intern mit minimalen Kopieroperationen zu implementieren (zero-copy).

Zusätzlich wurde ein `share_service` entwickelt, der den Zero-Copy-Austausch von Daten zwischen Applikationen auf demselben Knoten ermöglicht. Chunks können serialisiert und auf anderen Maschinen wiederhergestellt werden, was einen maschinenübergreifenden Datenaustausch erlaubt.

Darüber hinaus wurde eine FUSE-basierte Dateisystem-Präsentationsschicht implementiert, die den Zugriff auf IML-Storages über eine POSIX-konforme Dateisystem-Schnittstelle ermöglicht. Diese unterstützt Verzeichnisse, Dateien und symbolische Links mit Referenzzählung, File-Locking, Speichervorab-Allokation (`fallocate`) sowie dynamische Dateigrößenänderungen. Der Datei-Inhalt wird über ein austauschbares Backend realisiert, sodass sowohl lokale Implementierungen als auch verteilte MCS2-Storages als Speicher genutzt werden können.

Darüber hinaus wurden Kommandozeilen-Werkzeuge entwickelt, die sowohl zur Diagnose als auch zur Steuerung des Systems dienen. Für das `block_device` stehen unter anderem Befehle zur Verfügung, um Blöcke aufzulisten, deren Größe abzufragen, Daten zu lesen und zu schreiben sowie Blöcke zu entfernen. Das `iov_backend` bietet zusätzlich Werkzeuge zur Inspektion des Systemzustands, zur Auflistung von Storages und Collections, zur Abfrage der Datenverteilung auf die Storages sowie zum Import und Export von Daten. Beim Export und Import kann die eigentliche Dateiarbeit an die Storage-Server delegiert werden, was den Client entlastet und Kernel-Optimierungen wie `copy_file_range` nutzt. Diese Werkzeuge ermöglichen es, das Gesamtsystem auch ohne eigene Applikation zu testen und Fehler zu diagnostizieren.

Der gesamte Funktionsumfang der Client-Schnittstelle ist durch automatisierte Tests abgedeckt. Diese umfassen sowohl C++-basierte Integrationstests mittels Google Test als auch End-to-End-Tests, welche die Kommandozeilen-Werkzeuge in Multi-Prozess-Szenarien nutzen. Getestet wird über alle unterstützten Storage-Typen hinweg, sowohl synchron als auch asynchron.

AP3.d Implementierung BeeGFS User-Space Client (Leitung: GAUG/U) Im Rahmen des Arbeitspaketes AP3.d wurde die Machbarkeit einer BeeGFS Client Implementierung im User-Space untersucht und erfolgreich demonstriert. Ziel der Implementierung im User-Space war es, eine architektonische Grundlage für eine direkte und effiziente Integration des BeeGFS Client mit libIOVerbs zu schaffen. Eine solche Integration wäre im bestehenden BeeGFS Kernel Modul nur mit erheblichem Aufwand möglich gewesen. Ein weiterer Vorteil eines User-Space Dateisystem Ansatzes ist, dass viele Systemaufrufe und damit verbundene Kontextwechsel zwischen User- und Kernel-Space vermieden werden können, was konzeptionell die Vorraussetzung für eine Reduktion der Latenz pro Operation und damit eine erhöhte Operationsrate pro Thread schafft. Darüber hinaus ermöglicht dieser Ansatz eine flexiblere Weiterentwicklung unabhängig von Kernel-ABI-Änderungen.

Die Implementierung erfolgte in der Programmiersprache Rust. Rust bietet im Vergleich zu Sprachen wie C und C++ stärkere Garantien hinsichtlich Speicher- und Threadsicherheit, die bereits zur Kompilierzeit überprüft werden. Gleichzeitig bietet Rust viele moderne Sprachfeatures, die die Implementierung und Analyse von Applikationen vereinfachen. Dabei bleibt Rust nah an der Hardware und bietet viele Optimierungen zur Kompilierzeit und vermeidet dadurch zusätzliche Laufzeitkosten. Aufgrund dieser Eigenschaften sowie bestehenden Erfahrungen mit Rust in anderen BeeGFS-Komponenten wurde die Sprache als geeignet für die Systemimplementierung bewertet.

Initial wurde für den BeeGFS User-Space Client ein grundlegendes Design erarbeitet, das die wichtigsten Dateisystem Operationen (`open`, `close`, `read`, `write`) und andere, zusätzlich benötigte Funktionalitäten unterstützt. Auf dieser Basis wurde eine Implementierung realisiert, die die Dateisystem Operationen in BeeGFS RPCs umsetzt und an die BeeGFS Storage Server weiterleitet.

Zur Verbesserung des Lese- und Schreibdurchsatzes können, wie im etablierten BeeGFS Kernel Client, Dateien über mehrere BeeGFS Storage Server verteilt werden. Dateien werden dazu in gleich große Stücke (Chunks) zerlegt und nach einem vorher definierten Muster auf einer konfigurierbaren Anzahl von Servern verteilt. Für die Verbindung zu den Storage Servern wird das TCP Protokoll unterstützt.

Zur Validierung der Basisfunktionalität, noch vor vollständiger Spezifikation und Bereitstellung der libIOVerbs-API, wurde eine Bibliothek zum Abfangen von POSIX Dateisystem-Operationen verwendet. Dazu wurde zusätzliche Funktionalität, wie die Verwaltung von Dateideskriptoren im User-Space und deren Synchronisierung mit den Dateideskriptoren des Kernels, implementiert.

In der initialen Implementierung wurde das Metadatenmanagement bewusst auf ein minimales Mapping von Pfad zu Datei-ID beschränkt, um den Fokus auf die funktionale Machbarkeitsdemonstration zu legen.

Zusammenfassend wurde also im AP3.d die Machbarkeit einer BeeGFS Client Basisimplementierung im User-Space gezeigt. Diese Implementierung bietet eine Grundlage für ein abstrahiertes BeeGFS Backend, auf das sowohl libIOVerbs, als auch die bestehende, auf dem Abfangen von Systemaufrufen basierende Implementierung, aufsetzen können. Die weitergehende Integration in libIOVerbs wurde im Rahmen des AP2.c vorgenommen und die Funktionalität der Basisimplementierung sukzessive erweitert.

AP3.e BeeGFS Metadata Server Anpassungen (Leitung: GAUG/U) Im Arbeitspaket AP3.e wurde die Anpassung des BeeGFS Metadaten Servers als zentrale Metadatenkomponente für libIOVerbs im Lichte der projektspezifischen Anforderungen neu bewertet.

Zunächst wurde für die zentrale Metadatenkomponente hinter libIOVerbs in enger Zusammenarbeit mit den Projektpartnern ein Anforderungsprofil erstellt. Dabei wurden die folgenden Kernanforderungen herausgearbeitet:

- Ein zentraler Metadaten Service, der von vielen unabhängigen Instanzen von libIOVerbs effizient und parallel gelesen und geschrieben werden kann.
- Netzwerkbasierter Zugriff mit Unterstützung der in IO-Verbs definierten semantischen Anforderung bezüglich Konsistenz und Parallelität.
- Metadaten sollen anstatt des in POSIX üblichen hierarchischen Schemas (Dateisystembaum mit Ordnern und Unterordnern/Dateien) auch über eine Kombination verschiedener Tags, wie beispielsweise ein Experimentname, das Datum des Experiments und einer Iterationsnummer gefunden werden können.
- Eine einheitliche Interaktion mit derzeit und zukünftig unterstützten libIOVerbs Backend Implementierungen und die Möglichkeit, Backend spezifische Metadaten wie Informationen über Striping abzulegen, muss sichergestellt werden.

Die Analyse ergab, dass die Architektur des BeeGFS Metadaten Servers eng an eine hierarchische Ordnerstruktur und die damit verbundene strikte Eltern-Kind-Beziehung zwischen Ordnern und Einträgen in diesen gekoppelt ist, wie sie in POSIX Dateisystemen üblich ist. Der BeeGFS Metadaten Server verteilt auf dieser Basis Ordner auf verschiedene Server, was bei einer nicht-hierarchischen, tag-basierten Abfrage von Metadaten zu Ineffizienzen führen würde. Zusätzlich entsteht aus dieser Eltern-Kind-Beziehung eine Serialisierung von Operationen auf Objekten, die sich im Dateisystem im selben Ordner befinden, was mit einer flachen Metadaten-Struktur weitgehend vermieden werden kann.

Vor diesem Hintergrund wurde entschieden, ein wie im Rahmen von AP2.a beschriebenes neues Metadatenbackend auf Basis eines Key-Value-Stores mit optionalem Caching zu entwickeln. Die GAUG/U war hierbei an Design, Konzeption und API-Definition beteiligt und unterstützte die TUD bei der Entwicklung dieser Komponente.

Nach Bereitstellung der zentralen Metadatenkomponente wurde der BeeGFS User-Space Client an deren API angebunden, sodass Metadaten zentral gespeichert werden können. Über den Projektzeitraum wurden zwei verschiedene Entwicklungsstufen (lokal, basierend auf RocksDB und verteilt basierend auf ETCD mit optionalem RocksDB Cache) der Metadatenkomponente unterstützt, die größtenteils API kompatibel sind. Neben den global gültigen Metadaten können in der Metadatenkomponente auch BeeGFS spezifische Metadaten wie Striping Informationen abgelegt werden.

Mit den implementierten Features wurde das ausgearbeitete Anforderungsprofil erfüllt und damit AP3.e erfolgreich abgeschlossen.

AP3.f BeeGFS Komponenten Integration (Leitung: GAUG/U) Ziel von AP3.f war die Integration der in AP2.c, AP3.d und AP3.e entwickelten BeeGFS-Komponenten zu einer konsistenten, lauffähigen Gesamtlösung sowie die Bereitstellung einer reproduzierbaren Test- und Entwicklungsumgebung.

Hierzu wurde eine Docker basierte Umgebung zur Verfügung gestellt, mit der die benötigten Services gekapselt und damit mit geringem Konfigurationsaufwand lokal und auch in Cluster Umgebungen aufgesetzt und getestet werden können. Diese Umgebung ermöglicht es, das Gesamtsystem reproduzierbar aufzusetzen, Tests durchzuführen und neue Funktionen iterativ zu validieren.

Die Umgebung enthält neben einem unmodifizierten BeeGFS Management Server, an dem sich die Storage Server registrieren können, auch einen für dieses Projekt leicht modifizierten BeeGFS Storage Service, der die Daten, die über den User-Space Client abgelegt werden, in einem gesonderten Verzeichnis speichert. So können theoretisch sowohl über klassische BeeGFS Kernel Clients geschriebene Dateien, die von einem BeeGFS Metadatenservice verwaltet werden und Collections, die vom BeeGFS User-Space Client geschrieben werden und von der libIOVerbs Metadatenkomponente verwaltet werden, auf demselben Storage Service abgelegt werden. Dies erlaubt eine Koexistenz beider Zugriffspfade und stellt einen potenziellen Migrationspfad sicher.

BeeGFS verwendet als Einheit für das objektbasierte Striping sogenannte Targets, von denen ein Storage Service mehrere zeitgleich verwalten kann. Der User-Space Client kann über eine Konfigurationsdatei entsprechend konfiguriert werden. Hier werden mehrere Targets einem, durch eine IP-Adresse identifizierten, Storage Service zugeordnet. Dadurch ist es möglich, mehrere Storage Services mit jeweils mehreren Targets an das System anzubinden und damit Objekte über mehrere Maschinen und Massenspeicher zu verteilen, was eine Skalierbarkeit sicher stellt.

Für Funktionalitätstests wurden BeeGFS Backend spezifische Tests zur Verfügung gestellt, die ergänzend zu den allgemeinen Tests für libIOVerbs spezifische Szenarien abbilden. Außerdem wurde eine Integration in CMake entwickelt, die es erlaubt, den BeeGFS User-Space Client dynamisch als Abhängigkeit nachzuladen und automatisch zu bauen. Dies erleichtert die Integration in bestehende Build- und Testprozesse und unterstützt eine nachhaltige Weiterentwicklung.

Mit der erfolgreichen Integration aller Komponenten und der Bereitstellung einer reproduzierbaren Systemumgebung wurde AP3.f abgeschlossen. Die entwickelte Umgebung bildet die Grundlage für weiterführende Evaluierungen, Demonstratoren und potenzielle Folgeprojekte.

2.1.4 AP4: Campaign Storage (Leitung: GAUG)

Speicherplatz in den schnellen Scratch-Verzeichnissen von HPC-Clustern wird häufig aufgrund seiner Größe von Nutzern zur dauerhaften Speicherung von Simulationsdaten herangezogen. Um diese missbräuchliche Verwendung zu unterbinden, wird im Rahmen von Campaign Storage eine maximale Speicherfrist nach dem Ende einer Campaign festgelegt, um den Speicher im Anschluss für die weitere Verwendung durch andere Nutzer erneut freizugeben. Im Rahmen des vorliegenden Arbeitspaketes wird das in AP3 entwickelte MCS2 als Grundlage verwendet. Zur vereinfachten Anwendung durch Nutzer wird auf ein Workflow-System zurückgegriffen, um sowohl die Migration der Daten zwischen verschiedenen Speichersystemen (Storage Tiering) und den Ablauf der geplanten Simulationen abzubilden. Vorteilhaft an dieser Methodik ist auch die mögliche Übernahme des Workflows in Data Management Pläne und eine vereinfachte Reproduzierbarkeit von Simulationsstudien.

AP4.a MCS2 Workflow-Integration (Leitung: ITWM/GAUG) Im AP4.a wird MCS2 aus AP3 in ein Workflow-System integriert. Zum besseren Verständnis ist deshalb der Abschnitt in zwei Teile, jeweils einen für MCS2 und einen für das Workflow-System Snakemake, aufgeteilt.

MCS2 Das im AP3 entwickelte MCS2 ist eine Erweiterung des bereits vor dem Projekt bestehenden Speichersystems IML (Infinite Memory Layer) vom Fraunhofer ITWM. Zur Implementierung des MCS2 in anderen Anwendungen stehen dabei zwei Optionen, die Einbindung als Bibliothek bzw. durch den Aufruf statisch gelinkter Kommandozeilenbefehle, zur Verfügung. Im Falle der Workflow-Integration wird dabei auf die Kommandozeilenprogramme zurückgegriffen.

Snakemake Die Basis des im Arbeitspaket erarbeiteten Workflow-Systems ist das bereits existierende und u.a. in der Bioinformatik bekannte Open Source Workflow-Management-System Snakemake von Mölder et al. (2021). Ein weiterer Vorzug von Snakemake ist die bereits vorhandene Integration mit HPC-Schedulern und insbesondere mit dem im AP4.b erwähntem SLURM. Snakemake ermöglicht diese Integration durch ein Plugin-Interface, das neben Executors, wie z. B. für SLURM, auch eine Einbindung von anderen existierenden Speichersystemen (storage-plugins), Logging-Lösungen (logger-plugins) und Schedulern (scheduler-plugins) erlaubt.

Snakemake verwendet zur Darstellung von Workflows gerichtete azyklische Graphen (englisch DAG, directed acyclic graphs), wobei ein Knoten des Graphen eine Aufgabe (Task) darstellt, die durch eine Regel

(rule) in der sogenannten Snakefile-Datei beschreiben wird, wobei das folgende Listing 1 drei Beispiele für verschiedene Regeln zeigt:

Listing 1: Beispiel eines Snakemake Workflows

```
rule prepare:
    input: "path/to/experiment.conf"
    output: [inputs for 10 runs are created ... ]
    shell: "prepare␣inputs␣{input}"

rule executeTask:
    input: "one␣specific␣run␣input␣file", "some␣fixed␣generic␣file␣from␣a␣data␣pool"
    output: 3 products
    shell: "mpiexec␣...␣inputs␣{input}"

rule compareResults:
    input: outputs from all runs
    output: csv. ... paar pngs
    shell: "python␣"
```

Wie im Listing 1 zu sehen, verfügt jede Regel über einige Parameter, wie z. B. Input, Output oder Shell. Dabei entspricht 'input' den Eingabedateien, 'output' den Ergebnisdateien des Schritts und 'shell' der während der Anwendungen der Regel ausgeführten Kommandozeilenanweisung. Darüber hinaus gibt es weitere Möglichkeiten zur Code-Ausführung, wie z. B. die Ausführung von in der Snakefile implementierten Python-Funktionen.

Campaign Storage ist vergleichbar mit dem bereits existierenden Konzept von HPC-Workspaces, die Verzeichnisse auf Clustern mit einem Ablaufdatum darstellen. Dadurch wird es möglich, Speicher nur noch so lange zu belegen, wie es für die Durchführung des Projekts im HPC-System nötig ist und dadurch den vorhandenen Speicher effizienter zu nutzen. Nicht mehr benötigter Speicher, insbesondere im schnellen Verzeichnis, steht damit anderen Nutzern zur Verfügung. Zur Implementation des Konzepts in Snakemake wird das Attribut rresources" um ein weiteres Attribut campaigns" erweitert:

```
rule bwa\_map:
    resources:
        campaign: "100TB,10000files,scratch,2d"
    input:
        "input.txt"
    output:
        "output.txt"
    shell:
        "cp␣{input}␣{output}"
```

Im obigen Beispiel wird ein Speicher von 100TB mit zehntausend Dateien für zwei Tage auf scratch angefordert. Anders als gezeigt, ist es allerdings auch möglich, default resources anzugeben, die nicht exklusiv für einzelne Regeln sind.

Intern nutzt die Erweiterung von Snakemake diese Angaben, um den notwendigen Speicher bereitzustellen. Hierzu wird der Parser erweitert. Die Kommunikation mit dem Speichersystem erfolgt dann über die bereits angesprochenen Kommandozeilenbefehle. Alternativ wurde MCS2 als Speicher-Plugin in Snakemake implementiert. Dieses erlaubt nicht nur den direkten Snakemake-Code unverändert zu lassen, sondern ermöglicht auch die automatisierte Migration der Daten zwischen den verschiedenen Speichersystemen. Basis der Entwicklung ist das bereits vorhandene S3 storage plugin, das an MCS2 adaptiert wurde. Zunächst wurde ein Python-Wrapper für die verschiedenen CLI-Programme von MCS2 geschrieben. Diese wurden dann in den verschiedenen durch die Snakemake Storage Plugin Datenstrukturen vorgegebenen Funktionen verwendet. Zur Validierung der Implementierung wurde außerdem ein Testcase angelegt.

Mit der Integration von Snakemake und MCS2 wurde damit ein System vorgestellt, das die automatisierte Ausführung von Workflows auf MCS2 basiertem Speicher ermöglicht.

AP4.b SLURM Integration (Leitung: GAUG) Das von Jette and Wickberg (2023) entwickelte SLURM hat als Workload-Manager in Clustern weltweit eine starke Verbreitung und wird auch auf den Clustern der

GWDG eingesetzt. Zum erfolgreichen Einsatz von MCS2 ist deshalb eine Integration von SLURM zur Nutzung in Batch-Jobs unumgänglich. Die dabei zu lösenden Probleme beinhalten sowohl die Migration von Daten zwischen Speichersystemen, als auch die Überwachung und Steuerung des Ablaufs der verschiedenen Steps im Workflow.

Der Datentransfer kann neben dem in AP4.a beschriebenen Mechanismus auch direkt in SLURM durchgeführt werden. Die einfachste Integration würde dabei auf reine Dokumentation setzen, mittels derer der Nutzer sein Jobskript um die entsprechenden Schritte zur Datenmigration erweitert:

Listing 2: Manuelle Verwendung in Bash Jobskript

```
# Start the provider (with 1 thread) and save the connection information in a file
IML_IOV_PROVIDER_CONNECTABLE='/tmp/IML_IOV_PROVIDER_CONNECTABLE'
stdbuf -oL \
  ./iml_iov_backend_provider \
    "${IML_IOV_PROVIDER_ENDPOINT:?}" 1 | \
      tee "${IML_IOV_PROVIDER_CONNECTABLE:?}"
# Add Storages to the provider
./iml_iov_backend_storage_provider \
  "Just␣($(cat␣${IML_IOV_PROVIDER_CONNECTABLE:?}))" \
  'ip::tcp()' 1 \
  'ip::tcp()' 4 \
  "Heap␣(Limit␣$((2**30)))" \
  "Heap::Size::Max()" \
  "Heap::Size::Used()" \
  'Heap::Segment::Create␣(Nothing)' \
  'Heap::Segment::Remove()' \
  'Heap::Chunk::Description()' \
  'Heap::File::Read()' \
  'Heap::File::Write()'
# Create Collections
./iml_iov_backend_collection_create \
  "$(cat␣${IML_IOV_PROVIDER_CONNECTABLE:?})" \
  101 \
  $((10*2**20))
# Import data from a file
./iml_iov_backend_iov_write \
    "${IML_IOV_BACKEND_CONFIGURATION_FILE:?}" \
    "${IML_IOV_BACKEND_DATABASE_FILE:?}" \
    23 \
    "[0..$((8*2**30)))" \
    4 \
    $((128*2**20)) \
    6 \
    < DATA
```

Trotz der dadurch erzielten hohen Flexibilität wäre diese Vorgehensweise vermutlich nur für Poweruser möglich. Zur Vereinfachung der Nutzung durch weniger fortschrittliche Nutzer gibt es verschiedene Möglichkeiten, die Angaben direkt in der Snakefile zu hinterlegen. Das kann bereits das in AP4.a angesprochene Storage Plugin sein. Darüber hinaus können die Angaben als spezifische Ressourcen für jede einzelne Regel angegeben werden, wie bereits in Listing 2.1.4 gezeigt. Daneben ist es auch möglich, die Angaben als Default-Ressourcen des SLURM-executor-plugins zu hinterlegen:

```
executor: slurm
jobs: 2
default-resources:
    mem_mb: 200
    runtime: 100
    slurm_partition: 'medium'
```

Zur Integration in SLURM können die Angaben für die Bereitstellung des Speichers über die SLURM_EXTRA Umgebungsvariablen weitergegeben werden. Diese können im Anschluss in Prolog- und Epilog-Skripten

verarbeitet werden und den Speicher zu Beginn des Jobs bereitstellen bzw. am Ende des Jobs wieder freigeben. Eine weitere Alternative besteht darin, die Angaben aus dem Jobskript zu extrahieren. Auf dieses kann ebenfalls in Prolog- und Epilog-Skripten basierend auf der JOBID-Umgebungsvariable zugegriffen werden. Beispiele für derartige Skripte sind unten abgedruckt.

Listing 3: Beispiel Epilog-Skript

```
#!/bin/bash
#epilog
exec &> /home/cloud/prolog_slurmd.$$
set -x # To print out all commands

echo "epilog␣${SLURM_JOB_ID}␣$(hostname)"
/nfs/slurm/el8/slurm/24.05/install/bin/scontrol write batch_script ${SLURM_JOB_ID} /tmp/slurm-
    epilog-${SLURM_JOB_ID}.sh
data=$(/nfs/slurm/el8/slurm/24.05/install/bin/scontrol write batch_script ${SLURM_JOB_ID} -)

data2=$(grep "MCSE" /tmp/slurm-epilog-${SLURM_JOB_ID}.sh)
echo "$data" > /tmp/file-${SLURM_JOB_ID}
```

Listing 4: Beispiel Prolog-Skript

```
#!/bin/bash
#prolog
exec &> /home/cloud/prolog_slurmd.$$
set -x # To print out all commands

echo "prolog␣${SLURM_JOB_ID}␣$(hostname)"
/nfs/slurm/el8/slurm/24.05/install/bin/scontrol write batch_script ${SLURM_JOB_ID} slurm-prolog-${
    SLURM_JOB_ID}.sh
```

Um das Problem einer permanent aktiven Session auf Login-Nodes für die erfolgreiche Ausführung von Workflows zu lösen, wurde im Rahmen des Projekts eine dem SLURM-Daemon slurmctld vergleichbare Komponente snakeD entwickelt. Diese wird auf HPC-Infrastruktur-Knoten ausgeführt und unterliegt nicht den beschriebenen Einschränkungen. snakeD startet und überwacht dabei Snakemake Workflows und stellt dem Nutzer Monitoringdaten zur Verfügung. Zum Starten der Workflows wird dabei wiederum auf das SLURM-Executor-Plugin von Snakemake zurückgegriffen, wodurch eine nahtlose Integration in die bestehende Infrastruktur möglich wird. Darüber hinaus hat der Ansatz den Vorteil, dass auch weitere von Snakemake unterstütze Scheduler auf einfachste Weise integriert werden können.

AP4.c HPCSerA Anpassung (Leitung: GAUG) Normalerweise ist der Zugriff auf HPC-Cluster nur auf der Kommandozeile mittels SSH möglich. Für spezifische Anwendungen, wie zum Beispiel einem Web-Interface für numerische Berechnungen, ist dies allerdings nicht möglich, da der SSH-Schlüssel aus Sicherheitsgründen nur dem Nutzer selbst bekannt sein darf. Dieser Anwendungsfall kann das Paradigma Funktion as a Service (FaaS) mittels einer authentifizierten RESTful-API realisiert werden. Dabei erfolgt die Authentifizierung und Freigabe der verschiedenen Schritte über ein separates Web-Interface, wodurch sogar eine Vertraulichkeit der Daten gewährleistet werden kann. HPCSerA ist eine solche bei der GWDG von Bingert et al. (2021) entwickelte RESTful-API und erlaubt dabei das einfache Erstellen von Jobs auf dem Cluster. In diesem Rahmen könnte HPCSerA mit geringfügigen Modifikationen auch die Snakemake Jobs auf dem Cluster starten. Anders als im Antrag geplant, übernimmt HPCSerA nicht die Verarbeitung von DAGs aus Rechenjobs, sondern nutzt die vorhandene Funktionalität von Snakemake, das bereits um ein Storage-Plugin für MCS2 erweitert wurde. Zur einfacheren Benutzung wurde ein Python-basierter Kommandozeilen-Client entwickelt, über den Nutzer mittels einfacher Befehle die notwendigen Aktionen starten können. Dies sind im Einzelnen:

- **init:** Initialisiert eine neue Workflow-Ausführung oder fügt eine neue Ausführung einem bestehenden Workflow-Job hinzu. Dieser Befehl lädt die Snakefile hoch und löst die Vorbereitung des Arbeitsbereichs auf der Seite des Agents aus.

- **upload:** Lädt zusätzliche Eingabedateien hoch, die für die Workflow-Ausführung erforderlich sind, und bereitet sie für die spätere Verwendung im Workflow-Arbeitsbereich vor.
- **start:** Löst die Ausführung eines spezifischen Workflow-Laufs aus. Der Befehl dokumentiert Ausführungsparameter wie angeforderte Ressourcen und markiert den Lauf als ausführbar durch snakeD.
- **status:** Ruft den aktuellen Ausführungsstatus eines Workflow-Laufs ab und zeigt ihn an, einschließlich Metadaten und Ausführungsprotokolle, falls verfügbar.
- **delete:** Löscht workflowbezogene Dateien oder ganze Workflow-Jobs, die vom Dienst verwaltet werden, und löst entsprechende Bereinigungsaktionen auf der Agent-Seite aus, einschließlich optionaler Arbeitsbereichs-Freigabe.
- **list:** Listet Dateien auf, die im API-seitigen Job-Verzeichnis für einen bestimmten Workflow-Job gespeichert sind.

Optional enthält das Design auch die Möglichkeit, HPCSerA als Alternative zu SLURM als weiteres HPC-backend in Snakemake zu implementieren. In Fällen ohne nötigen Upload auf den Cluster werden die Campaign-Befehle entweder als Teil des Snakemake Workflows oder des SLURM-Jobscripts umgesetzt. Zukünftige Erweiterungen könnten alternativ auf den ebenfalls bei der GWDG u.a. von Decker (2025) entwickelten Scalable Artificial Intelligence (AI) Accelerator 2 (SAIA-2) zurückgreifen.

Damit ist die effektive Nutzung des in AP4 entwickelten Workflow-Systems auf dem Cluster der GWDG oder anderen SLURM basierten Clustern gewährleistet.

2.1.5 AP5: Anwendungsportierung (Leitung: DLR)

AP5.a Portierung der IO500 (Leitung: GAUG) Die IO500 von Kunkel et al. (2017) verwendet die Benchmarks aus der weit verbreiteten IOR-Benchmark-Suite und der pfind Bibliothek. Die IOR-Benchmark-Suite berücksichtigt dabei sowohl das Best-Case (easy) als auch Worst-Case (hard) Szenario in Hinblick auf Zugriffsmuster bei der Bestimmung der Leistungskennzahlen. Da die IOR-Benchmarks allerdings Beschränkungen durch Entscheidungen der IOR-Entwickler unterliegen, ist die abstrakte IOR-API (aiori) nicht geeignet, alle Möglichkeiten der in den IOVerbs definierten Semantiken abzubilden. Zur flexibleren Implementierung des IOVerbs-Backends wurde deshalb entschieden, auf die Realisierung entsprechend der aiori API zu verzichten und stattdessen eine semantisch kompatible Neu-Implementation der Phasen mittels IOVerbs umzusetzen. Damit wird die maximal mögliche Performanz erreicht, was eine realistischere Einschätzung des Potenzials der libIOVerbs-API ermöglicht. Aufgrund der noch nicht abgeschlossenen Implementierungen der IOVerbs C-API wird in einem ersten Schritt die API ohne entsprechende volle Funktionalität verwendet. Stattdessen wurde ein Mockup entwickelt, das sicherstellt, dass der entwickelte Code bereits erfolgreich mit der libIOVerbs-API kompiliert werden kann. Dieses wurde auch auf der ISC 2024 im Rahmen eines BOF präsentiert und im Quellcode auf GitHub Kunkel (2024) veröffentlicht. Durch das Mockup kann die Implementierung zunächst von anderen Arbeitspaketen entkoppelt und der allgemeine Projekt-Fortschritt beschleunigt werden. Im nächsten Schritt wird die vollständige Implementierung der IO500 entsprechend des erstellten Designs erfolgen. Im Rahmen der Design-Erstellung wurde festgestellt, dass die IOVerbs-Spezifikation in Bezug auf Metadaten erweitert werden muss. Da die initiale Spezifikation noch keine verpflichtenden Key-Value-Paare definiert, wurden im Design der IO500 Portierung applikationsspezifische Paare angenommen.

Auf Basis des in diesem und nächsten Abschnitts beschriebenen Designs ist eine Implementierung der IO500 mittels der libIOVerbs-API möglich und der Meilenstein 5.2 entsprechend erreicht.

Die Mockup-Implementierung setzt sich wie die originale Implementierung aus einer Kombination der Benchmarks aus der IOR-Suite und pfind zusammen. Die folgenden Abschnitte beschreiben die dabei in beiden Modulen implementierten Konzepte und Funktionen.

IOR-Mockup Das IOR-Mockup erfasst Tests aus zwei verschiedenen Quellen: IOR selbst und mdtest. Dabei messen mdtest die Leistungsfähigkeit der Speichersysteme für Metadaten-Operationen und IOR den Durchsatz der eigentlichen Daten-Operationen.

Im Sinne von IOR bezieht sich die Unterscheidung zwischen einfach (easy) und schwierig (hard) auf Muster beim Dateizugriff. Einfache Dateizugriffsmuster sind durch wenige große zusammenhängende

Speicherbereiche, auf die sequenziell zugegriffen wird, gekennzeichnet. Schwierige Zugriffsmuster beinhalten dagegen zufällige Zugriffe auf kleine, verstreute Speicherblöcke. Im Rahmen der IOR-Tests werden diese Muster für lesende und schreibende Zugriffe evaluiert. Hierzu wird zunächst ein Workspace und anschließend in diesem eine Collection erzeugt mit der die Tests durchgeführt werden. Im Anschluss wird lesend bzw. schreibend auf die erzeugte Collection zugegriffen und die hierfür benötigte Zeit gemessen. Die verwendeten libIOVerbs-API-Funktionen sind `iov_batch_create`, `iov_batch_start`, `iov_batch_free`, `iov_wait`, `iov_workspace_create`, `iov_collection_create` und `iov_write` bzw. `iov_read`. Als Erstes werden die Schreib-Tests ausgeführt und anschließend die Lese-Tests.

Wie bereits im letzten Abschnitt geschrieben, prüfen die Benchmarks in mdtest die Leistungsfähigkeit der Metadaten-Operationen von Speichersystemen. In diesem Fall bezieht sich einfach (easy) und schwierig (hard) auf Metadaten-Operationen von leeren Dateien bzw. solchen mit Nutzdaten. Dabei werden zunächst der Workspace und die enthaltenen Collections im ersten Schritt erstellt. Im nächsten Schritt werden die Metadaten für alle erstellten Collections abgefragt und angezeigt. Im Anschluss werden die Collections mit Inhalt gelesen. Zum Abschluss werden alle Collections und der jeweilige Workspace erneut gelöscht. Zur Prüfung der Metadaten-Operationen wird von jedem einzelnen Schritt die benötigte Zeit und darauf basierend die Leistungskennzahlen ermittelt. Die verwendeten libIOVerbs-API-Funktionen sind `iov_batch_create`, `iov_batch_start`, `iov_batch_free`, `iov_wait`, `iov_workspace_create`, `iov_workspace_load`, `iov_workspace_search`, `iov_collection_create`, `iov_collection_lookup`, `iov_collection_search` und `iov_write` bzw. `iov_read`.

Die Implementierung der einzelnen Phasen wird hier am Beispiel von easy_write verdeutlicht. Die Funktion erstellt leere Collections mit process-spezifischen Metadaten-Key-Value-Paaren. Dabei muss zwischen Metadaten-Key-Value-Paaren für den Workspace und die Collections unterschieden werden. Zusätzlich werden Controls definiert, die die I/O Semantiken der Anwendung in IOVerbs abbilden. Im parallelen Test wird dabei der Workspace vom Prozess mit Rank 0 durch die Funktion `iov_workspace_create` angelegt. Die anderen Prozesse öffnen den Workspace mit `iov_workspace_load`. Im Rahmen der IOVerbs werden Dateisystemoperationen in Batches gesammelt ausgeführt. Jeder Prozess erstellt einen Batch mittels `iov_batch_create`. Während dem Benchmarklauf werden die Operationen `iov_collection_create`, `iov_collection_close` und `iov_collection_lookup` im Batch registriert. Dieser wird im Anschluss mit `iov_batch_start` asynchron gestartet. Die Funktion `iov_wait` überprüft regelmäßig, ob alle Operationen im Batch abgeschlossen wurden. Im Anschluss wird der Programmfluss fortgesetzt und der Speicher des Batches freigegeben. Auf Basis der benötigten Zeit lässt sich die Geschwindigkeit der Erstellung der Collections ermitteln. Nach Ablauf der Stonewall-Time beenden alle Prozesse die Erstellung von Collections und ermitteln das Maximum der erstellten Collections der verschiedenen Prozesse. Im zweiten Schritt erstellen die verschiedenen Prozesse die verbleibenden Collections um eine identische Anzahl in allen Prozessen zu erreichen. Die Vorgehensweise ist identisch mit der vorher beschriebenen Prozedur. Zum Abschluss wird der Workspace von allen Prozessen mit `iov_workspace_close` geschlossen und die Laufzeit, Erstellungsrate der Collections und die Anzahl der Collections bestimmt und an den Benchmark zurückgegeben.

pfind-Mockup Das Mockup von pfind erstellt zunächst eine größere Anzahl von Collection-Objekten mit mehreren Key-Value-Metadaten-Paaren. Diese können im Anschluss für die Suche verwendet werden. Zur Umsetzung der Konzepte wurden die Datenstrukturen iov_workspace und iov_collection im Rahmen der Implementierung erweitert, um im lokalen Arbeitsspeicher die Auflistung der Collections im Workspace und die Metadaten der Collections emulieren zu können. Die verwendeten libIOVerbs-API-Funktionen sind `iov_batch_create`, `iov_batch_start`, `iov_batch_free`, `iov_wait`, `iov_workspace_create`, `iov_workspace_load`, `iov_workspace_search`, `iov_collection_create`, `iov_collection_lookup` und `iov_collection_search`.

Obwohl alle Funktionen vollständig implementiert wurden, bestehen bei der Kopplung der C-API mder entwickelnden Metadaten-Komponente noch Integrationsprobleme, wodurch eine Ausführung des kompletten IO500-Benchmarks zum Projektende nicht mehr erreichbar war und damit in einem möglichen Folgeprojekt abgeschlossen werden müsste.

AP5.b Generalisierung der I/O in FSDM (Leitung: DLR) Im Rahmen von AP5.b wurden die typischen I/O-Muster für Simulationsdaten aus der numerischen Strömungsmechanik identifiziert und in einer generischen Form, unabhängig von den spezifischen Anwendungen FSDM und CODA, dokumentiert (siehe auch

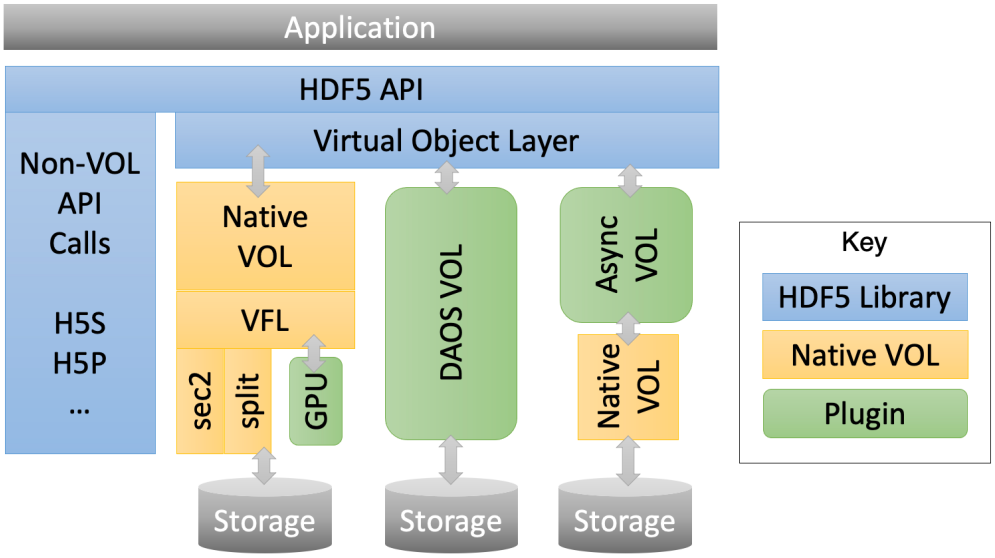

Abbildung 6: Das Virtual Object Layer (VOL) und das Virtual File Layer (VFL) im HDF5 Datenmodell © The HDF Group, `https://www.hdfgroup.org/wp-content/uploads/2022/02/VOL_tutorial_feb_2022.pdf`

AP1.a). In dem Prozess wurden auch die bestehenden I/O Schnittstellen von FSDM auf ihre Relevanz und Eignung für die in par:ap1d zu entwickelnde High-Level I/O-API für Strömungssimulationsdaten hin untersucht. Dabei hat sich das HDF5-Dateiformat (The HDF Group (2026)) durch seine umfangreiche C-API, die Verfügbarkeit von High-Level-C++-APIs sowie sein selbstbeschreibendes Datenmodell als besonders geeignet erwiesen. Durch seine etablierte Nutzung im DLR sind viele Anwender zudem bereits mit HDF5-Dateien und Werkzeugen wie `h5dump` oder `h5diff` vertraut, was den Einstieg erleichtert.

AP5.c und AP5.d Portierung von FSDM für direkten Speicherzugriff und IOVerbs (Leitung: DLR) Basierend auf der Entscheidung für das HDF5-Format wurde nach Möglichkeiten gesucht, wie eine HDF5-Datei auf IML/MCS2 beziehungsweise die libIOVerbs-Bibliothek abgebildet werden kann. Die HDF5-API stellt für diesen Zweck eine “Virtual Object Layer”- und eine “Virtual File Layer”-Plugin-API (siehe Abbildung 6) bereit. Diese beiden Schnittstellen wurden im Rahmen des Projekts zunächst auf ihre Eignung zur Anbindung von IML/MCS2 beziehungsweise der libIOVerbs-Bibliothek hin untersucht.

Das Virtual Object Layer (VOL) und das Virtual File Layer (VFL) erlauben zu beeinflussen, wie Objekte im HDF5 Datenmodell (Gruppen, Datensätze und Attribute) auf unterschiedliche Speichersysteme abgebildet werden (siehe Abbildung 7). Die *Virtual Object Layer (VOL)*-Schnittstelle dient primär der Abbildung von Daten im HDF5-Datenmodell (d.h. einer Hierarchie von Gruppen, Datensätzen und Attributen) auf ein bestimmtes Dateiformat. Dies ermöglicht es, Daten in einem anderen Dateiformat – oder sogar in nicht dateibasierten Systemen wie Datenbanken – abzulegen, während Anwendungen weiterhin über die gewohnte HDF5-API und das HDF5-Datenmodell auf diese zugreifen können. Hierzu werden alle Funktionsaufrufe, die auf Objekte im HDF5-Datenmodell zugreifen, abgefangen und anstelle einer nativen VOL-Implementierung an Funktionen weitergeleitet, die von einem externen VOL-Connector bereitgestellt werden.

Andererseits bietet das *Virtual File Layer (VFL)* die Möglichkeit, durch die Verwendung verschiedener Virtual File Driver (VFD) (z.B. “sec2” in Abbildung 6) zu beeinflussen, wie Daten bei Verwendung der nativen VOL-Implementierung auf das unterliegende Speichersystem abgebildet werden. Hierzu sind bereits mehrere VFDs in einer Standard-HDF5-Installation enthalten. Dazu zählen unter anderem der standardmäßig genutzte “sec2”-VFD, der sequentiell unter Verwendung von POSIX-I/O auf das Dateisystem zugreift sowie der “MPI-IO”-VFD, der parallel unter Verwendung von MPI-IO auf das Dateisystem zugreift.

Da nichts gegen die Verwendung des HDF5-eigenen Dateiformats sprach, wurde festgestellt, dass eine Implementierung eines VOL-Plugins zur Erreichung der Projektziele nicht erforderlich ist. Stattdessen wurde eine Anbindung von IML/MCS2 und der libIOVerbs-Bibliothek an die im Projekt entwickelte High-Level-API für Strömungssimulationsdaten (siehe Abschnitt zu AP1.d) mittels jeweils eines VFD vorgenommen.

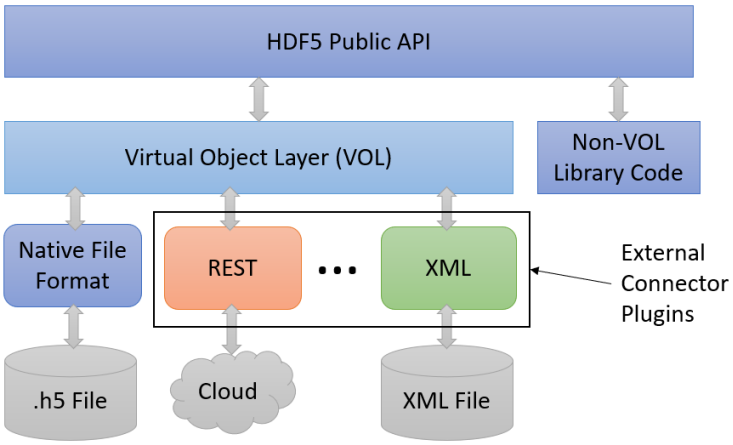

Abbildung 7: Beziehung zwischen Virtual Object Layer (VOL) und der applikationsseitigen HDF5 API © The HDF Group, `https://support.hdfgroup.org/documentation/hdf5/latest/vol_architecture.png`

Da das Durchreichen von I/O-Semantiken ein zentrales Projektziel darstellt, musste insbesondere bei der Implementierung des VFD für IOVerbs eine Möglichkeit geschaffen werden, semantikbeschreibende Parameter von der Anwendungsseite über den File Driver an die libIOVerbs-Bibliothek weiterzureichen. Die HDF5-API stellt hierfür sogenannte „File Creation“, „File Access“ und „Data Transfer“ Property Lists zur Verfügung. Dabei handelt es sich um C-Strukturen, deren Lebensdauer (Speicherallokation und -deallokation) von der HDF5-Bibliothek verwaltet wird und auf die über opake IDs zugegriffen wird. Diese Property Lists sind grundsätzlich VFD-spezifisch: Während die HDF5-Kernbibliothek lediglich die Property-List-Infrastruktur bereitstellt, werden Inhalt, Bedeutung und Auswertung der darin abgelegten Parameter ausschließlich vom jeweils verwendeten VFD definiert. Eine Anwendung kann somit für einen bestimmten VFD eine passende Property List erzeugen, diese mit geeigneten, VFD-spezifischen Parametern konfigurieren und anschließend deren ID bei Aufrufen von HDF5-API-Funktionen übergeben.

Beispielsweise kann eine File Access Property List bei Aufrufen von `H5Fcreate` oder `H5Fopen` angegeben werden. Die dazu übergebene ID wird dabei unverändert durch die internen Schichten der HDF5-Bibliothek an den gewählten VFD weitergereicht und steht schließlich in dessen `open`-Callback zur Verfügung. An dieser Stelle kann der VFD die zugehörige File Access Property List von der HDF5-Bibliothek abfragen und die darin enthaltenen VFD-spezifischen Parameter gemäß seiner eigenen Semantik interpretieren. Im Fall des IOVerbs-VFD können somit in der File Access Property List die im Absatz zu AP1.b beschriebenen Controls zur Konfiguration der I/O-Semantiken abgelegt, im `open`-Callback interpretiert und bei Aufrufen der IOVerbs-Routinen `load/create_workspace` beziehungsweise `load/create_collection` übergeben werden.

Bei der konkreten Ausgestaltung der Implementierung wurde konsequent auf eine möglichst hohe Wiederverwendbarkeit und Anwendungsagnostik geachtet. Die entwickelten VFDs sind vollständig unabhängig von einem spezifischen Anwendungsszenario und hängen ausschließlich von der libIOVerbs-Bibliothek, MPI sowie der HDF5-C-API ab. Insbesondere trifft der VFD für IOVerbs keinerlei domänenspezifische Annahmen über die I/O-Semantiken, sondern implementiert lediglich die für den Dateizugriff erforderlichen Schnittstellen gemäß der HDF5-VFD-API unter Berücksichtigung der von Anwendungsseite in Property Lists spezifizierten semantischen Controls.

Ergänzend dazu wurde ein C++-basierter IOVerbs-Treiber in Form einer eigenständigen Klasse implementiert, der für die Verwendung mit der High-Level-HDF5-C++-API *h5cpp* ausgelegt ist. Auch diese Komponente ist generisch gehalten und kapselt die Interaktion mit dem IOVerbs-VFD, ohne an einen konkreten Anwendungsfall gebunden zu sein. Damit kann der VFD einfacher in C++-Anwendungen verwendet werden.

Zur Unterstützung unterschiedlicher Speicherbackends wurde in der High-Level-API für die Strömungsmechanik (siehe Abschnitt zu AP1.d) eine Backend-Schnittstelle geschaffen, über die der Anwender zwischen verschiedenen Backends wählen kann, beispielsweise einem MPI-IO-Backend oder dem IOVerbs-Backend. Letzteres baut auf dem zuvor beschriebenen C++-Treiber auf. Auf Grundlage der in AP1.a identifizierten domänenspezifischen semantischen Anforderungen an das Datei-I/O werden in der Implementie-

rung des IOVerbs-Backends die konkreten I/O-Semantiken für den Anwendungsfall der Strömungsmechanik festgelegt und intern an die generische Schnittstelle des C++-IOVerbs-Treibers übergeben.

Die strikte Trennung in generische (VFD in C und zugehöriger C++-Treiber) und domänenspezifische Schichten (IOVerbs-Backend in der High-Level-API für Strömungssimulationsdaten) hat den Vorteil, dass sowohl der VFD als auch der zugehörige C++-Treiber in anderen Domänen wiederverwendet werden können. Insbesondere erlaubt die generische Umsetzung des VFDs die Nutzung mit bestehenden HDF5-Werkzeugen wie `h5dump` oder `h5diff` zum Inspizieren oder Vergleichen der Inhalte von HDF5-Dateien, die mittels IOVerbs in IML oder BeeGFS abgelegt wurden. Andererseits wird der Endnutzer durch die Festlegung auf für Strömungssimulationsanwendungen ausgelegte I/O-Semantiken entlastet, da keine detaillierten Kenntnisse über verschiedene semantische Anforderungen an Datei-I/O und deren Auswirkungen auf Korrektheit und Performanz erforderlich sind.

Entsprechend wurde auch das benutzerseitige Python-Interface von FSDM so erweitert, dass der Wechsel zwischen unterschiedlichen HDF5-Backends – einschließlich des IOVerbs-basierten Zugriffs – mit minimalem Anpassungsaufwand möglich ist. Für den Anwender beschränkt sich diese Entscheidung auf wenige Zeilen Code, während die Auswahl der passenden I/O-Semantiken sowie deren Übergabe an den Virtual File Driver vollständig durch die Bibliothek übernommen wird. Das folgende Codebeispiel zeigt, wie der Anwender ein Backend auswählen und in der High-Level-API für Lese- und Schreiboperationen einsetzen kann.

```python
from FSDataManager import FSClac, FSMesh

import fsmeshio
from fsmeshio.backends.mpiio import MPIBackend
from fsmeshio.backends.iov import IOVBackend

clac = FSClac()
mesh = FSMesh(clac)

iov_backend = IOVBackend(clac)
# Alternativ:
# mpi_backend = MPIBackend(clac)

fsmeshio.read(mesh, "input_mesh.h5", backend=iov_backend) # Oder mpi_backend
# ... Simulation ...
fsmeshio.write(mesh, "simulation_result.h5", backend=iov_backend) # Oder mpi_backend
```

AP5.e Optimierung von FSDM (Leitung: DLR) Basierend auf den Analysen der I/O Zugriffsmuster von FSDM in AP1.a wurden verbesserte Import- und Exportroutinen entwickelt, die das parallele Dateisystem von HPC Systemen effizienter ausnutzen. Die Performanzgewinne dieser Implementierung gegenüber der bestehenden Implementierung in FSDM wurden mithilfe eines für die Luftfahrtforschung relevanten Realweltdatensatzes nachgewiesen (siehe Abbildung 8).

Der verwendete Datensatz basiert auf dem CRM-Netz (Common Research Model) mit Verfeinerungslevel E vom 4. High Lift Prediction Workshop[32] und umfasst etwa 630 Mio. Knoten sowie 780 Mio. Volumenzellen. Zusätzlich zum Netz sind die Ergebnisse einer typischen RANS-SAneg-Simulation enthalten, bei der sechs `double` Variablen pro Volumenzelle abgelegt werden. Insgesamt ergibt sich so eine Gesamtdatenmenge von rund $100\,\mathrm{GB}$.

Die Simulationen wurden auf dem vom DLR betriebenen Cluster CARO durchgeführt. Hierbei wurden SSD-basierte Object Storage Targets (OSTs) des Lustre-Dateisystems verwendet. Pro Rechenknoten kamen 32 MPI-Tasks zum Einsatz. Die Daten wurden über 16 OSTs mit einer Stripe-Größe von $2\,\mathrm{MiB}$ verteilt. Der HDF5-spezifische *alignment* Parameter wurde ebenfalls auf $2\,\mathrm{MiB}$ gesetzt, um die I/O-Performanz zu optimieren.

Aus den abgebildeten Kurven wird deutlich, dass die ursprünglich verwendeten Import- und Exportroutinen (orange) nicht mit der Anzahl der Rechenknoten skalieren. Die neu entwickelten Routinen (blau) zeigen hingegen bis etwa 32 Rechenknoten ein nahezu ideales Skalierungsverhalten und erreichen auch

[32] `https://web.archive.org/web/20250601022050/https://hiliftpw.larc.nasa.gov/Workshop4/grids_downloads.html`

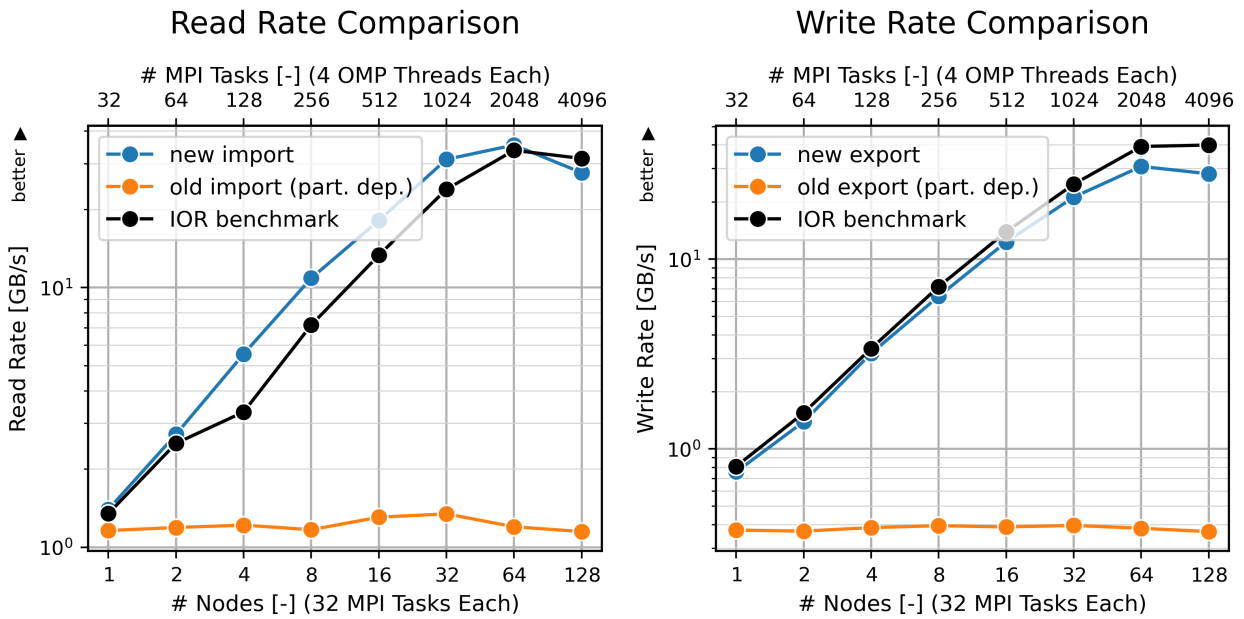

Abbildung 8: Vergleich von Lese- und Schreibraten zwischen bei Projektstart vorhandenen naiven Import- und Export Implementierungen und für die verwendeten Zugriffsmuster auf parallelen Dateisystem optimierten Implementierungen.

darüber hinaus eine vergleichbare Leistung wie ein klassischer IOR-Benchmark (schwarz), der als Richtwert für die theoretisch erzielbaren Lese- und Schreibraten des verwendeten Speichersystems dient. Der Rückgang der Lese- und Schreibraten ab 64 Knoten ist primär auf die Menge an geschriebenen Daten pro Datensatz und Prozess zurückzuführen, die in diesem Bereich die Größenordnung der Stripe-Size und des HDF5 alignments erreicht und somit zu einem zunehmenden Speicheroverhead führt. Erste Tests haben gezeigt, dass bei noch größeren Datenmengen, mit denen im Exascale-Zeitalter zu rechnen ist, nahezu ideale Skalierung bis zum hier getesteten Maximum von 4096 Prozessen erreicht werden kann.

2.1.6 AP6: Evaluation (Leitung: GAUG/U)

AP6.a Erstellung von Microbenchmarks (Leitung: GAUG/U) Schon während der Implementierungsphase wurden von allen Projektpartnern Tests entwickelt und gepflegt, um eine effiziente Entwicklung zu gewährleisten und Fehler und potenzielle Bottlenecks im eigenen Code, aber auch Ineffizienzen und Probleme im libIOVerbs-API frühzeitig erkennen und beheben zu können. Die Erkenntnisse daraus sind über die gesamte Projektlaufzeit als Verbesserungen in die einzelnen Komponenten eingeflossen.

Für IML wurden Microbenchmarks für *put* und *get* Operationen durchgeführt. Genutzt wurde ein 10Gbit/s Ethernet Netzwerk, wobei das Netzwerk für diese Tests auch von anderen Nutzern des Clusters genutzt wurde. Nur die Nodes selbst wurden exklusiv genutzt. Der Provider wurde immer mit sechs Threads ausgeführt und der Client mir vier Threads. Für jedes Kommando umfasst eine Messeinheit folgende Ausführungsschritte:

- Serialisieren und absenden der Kontrollnachricht vom Client zum Provider via *TCP/IP*.
- Empfangen und deserialisieren der Kontrollnachricht.
- Kopieren der Daten über Ethernet.
- Serialisieren und absenden des entsprechenden Rückgabewerts der *put* und *get* Funktion vom Provider zum Client via *TCP/IP*.
- Empfangen und deserialisieren des Rückgabewerts.

Abbildung 9 zeigt die erreichte Bandbreite bei Paketgrößen bis zu 8MB. Hier erreicht IML bei ausreichend großen Paketen eine Netzwerkauslastung von über 90%.

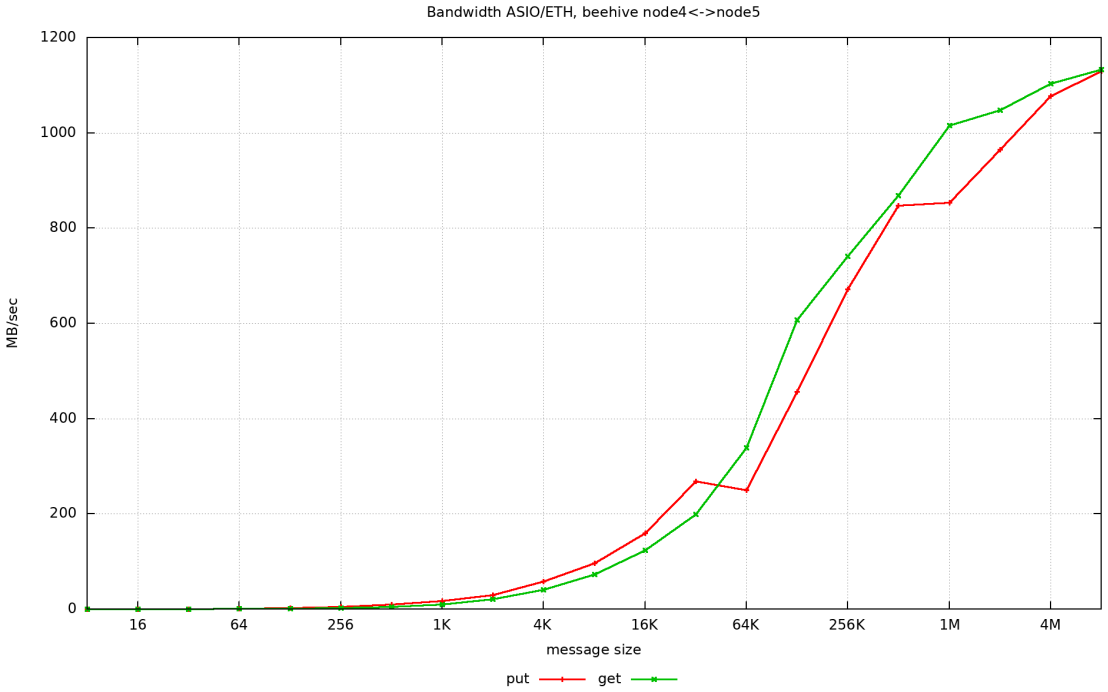

Abbildung 9: IML Benchmark: Bandbreite

Abbildung 10 zeigt die Latenz bei Paketgrößen bis zu 16KB. Bei größeren Paketen steigt die Latenz proportional zur Paketgröße, was gewünscht ist. Daraus lässt sich schließen, dass der Berechnungsoverhead der oben genannten Schritte (ohne das Kopieren) konstant bleibt. Somit ist dieser unabhängig von der Größe der übertragenen Daten. Der Overhead, den IML zusätzlich zum Datentransport benötigt, ist bei *get* weniger als ca. 120μs und bei *put* weniger als ca. 80μs.

Beide Werte können vorerst als sehr gut eingestuft werden. Daher wurden keine Vergleichsmessungen mit anderen Verfahren oder Tools durchgeführt. Als sehr rudimentären Vergleichswert kann man *ping* heranziehen: Der geringste durchschnittliche Wert, den *ping* für beliebig kleine Pakete zwischen den gleichen IP-Adressen erreichen konnte, liegt bei ca. 150μs. Somit ist die Latenz für *IML-put* und *IML-get* niedriger als die für *ping*.

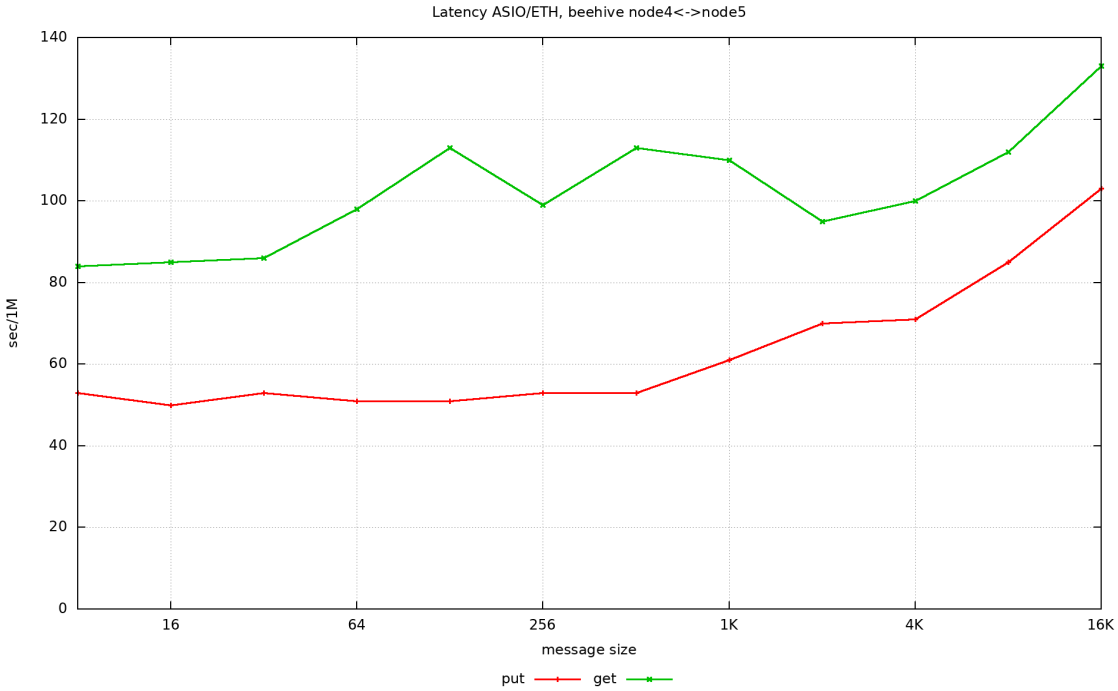

Abbildung 10: IML Benchmark: Latenz

Wie bereits in AP5.a beschrieben hat die Portierung der IO500 auf die IOVerbs nicht mehr die volle Funktionalität erreicht.

Das DLR hat einen Ende-zu-Ende Benchmark für die Anwendungsdomäne der numerischen Strömungssimulation entworfen. Bei diesem wird ein Simulationsnetz aus FSDM heraus zunächst in eine HDF5 Datei exportiert und im Anschluss wieder importiert. Dabei wird die in Abschnitt AP1.d beschriebene High-Level IO-API verwendet. Zum Lesen und Schreiben der Daten in die verschiedenen Speicherbackends (IML, BeeGFS, Cluster-Eigenes Lustre Dateisystem) wird der in Abschnitt AP5d beschriebene Virtual File Driver für IOVerbs bzw. der von HDF5 bereitgestellten MPI-IO Virtual File Driver verwendet.

AP6.b Deployment (Leitung: GAUG/U) Im Rahmen von AP6.b wurden die entwickelten Komponenten in unterschiedlichen Konfigurationen gemeinsam deployt und unter realistischen Bedingungen auf Clustersystemen ausgeführt.

Um auf die verschiedensten Umgebungen vorbereitet zu sein und sowohl lokal als auch im größeren Maßstab effizient Systeme aufbauen, testen und benchmarken zu können, wurde im Projekt für das Deployment sowohl der Bibliotheks-, Metadaten- und Applikations- als auch der Backendkomponenten konsequent auf Containerisierung (Docker, Podman, Apptainer) gesetzt. So konnten Komponenten sowohl isoliert in CI Umgebungen, als auch integriert auf Clustersystemen ohne erhöhten Installationsaufwand der Projektpartner und ohne weitreichende Privilegien ausgeführt werden. Letzteres ist insbesondere für extern verwaltete Systeme, auf denen nicht ohne weiteres höhere Privilegien erlangt werden können, notwendig. Zusätzlich erlaubte diese Kapselung in Containern einen einfachen Austausch kompletter Umgebungen in einem eingefrorenen Softwarezustand, was die Fehlersuche und -behebung durch vereinfachte Iterationsfähigkeit erheblich erleichtert hat.

Für die Backends waren üblicherweise mehrere Container-Images nötig, um komplette verteilte Systeme starten zu können (Meta-Image und Storage-Image), während die Bibliothekskomponenten zusammen mit der jeweiligen Applikation in einem Container-Image installiert und entsprechende Container auf mehrere Client Maschinen ausgeführt werden konnten. Abbildung 11 veranschaulicht wie das Deployment mittels Containern auf einem Clustersystem aussehen kann. Der Clusteraufbau in diesem Bild orientiert sich dabei am Aufbau der DLR-Systeme CARO und CARA: An einem Hochgeschwindigkeitsnetzwerk sind sowohl Rechenknoten (oben) als auch das clustereigene verteilte Dateisystem (Lustre im Fall von CARO und CARA) unten angeschlossen. Letzteres besteht aus Metadatenservern, die Metadaten von im Dateisystem

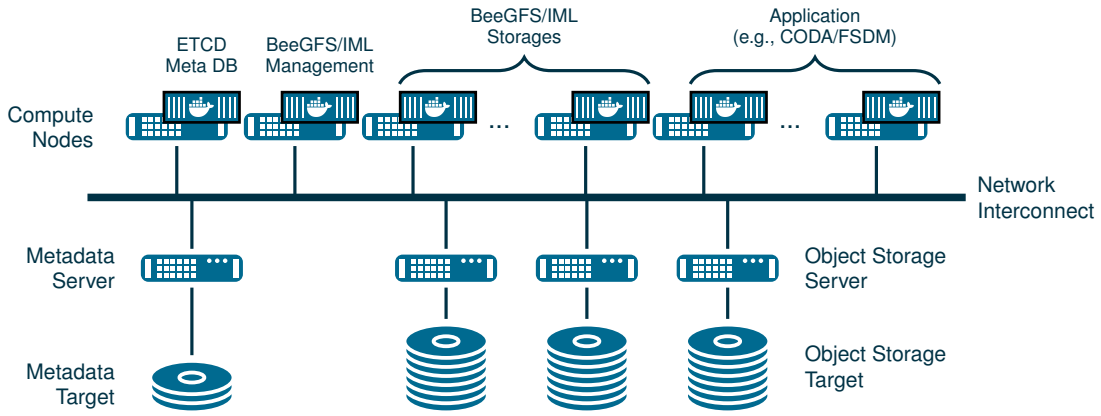

Abbildung 11: Deployment Setup auf Cluster Systemen

abgelegten Dateien auf Metadatentargets ablegen und Object Storage Servern, die die tatsächlichen Datei-Inhalte auf Object Storage Targets ablegen. Wie bereits oben erwähnt, ist es Nutzern solcher fremdverwalteten Systeme nicht ohne weiteres möglich, alternative Dateisysteme wie BeeGFS oder IML aufzusetzen.

Hier kommt die verwendete Containerstrategie zum Tragen: Analog zu den dedizierten Lustre Knoten werden zusätzlich zu den Rechenknoten, die sowieso für die Anwendung alloziert werden, weitere Standard-Rechenknoten des Clusters angefordert, auf denen ein Dateisystem ad-hoc aufgebaut wird. Dazu wird auf einem Rechenknoten ein containerisierter ETCD Metadatenserver gestartet. Auf weiteren Knoten werden BeeGFS oder IML Management Container ausgeführt und ein oder mehrere zugehörige Storage-Container ausgeführt. Nach erfolgreichem Aufbau aller Komponenten des Speichersystems werden auf den verbleibenden Rechenknoten Anwendungscontainer gestartet, in denen sich etwa eine FSDM/CODA Installation befinden kann, die gegen die libIOVerbs-Bibliothek gelinkt ist und die über eine IOVerbs spezifische Konfigurationsdatei die benötigten Informationen zum Zugriff auf die Knoten des ad-hoc Speichersystems erhält.

Die Netzwerkkommunikation zwischen den Rechenknoten ist ohne aufwendige Konfiguration möglich, da Applikationsports der Speichersysteme von der verwendeten Containerruntime (Podman, Apptainer, etc.) auf Ports der Rechenknoten abgebildet werden. Das vollständige Setup des Speichersystems und der Start der Anwendung innerhalb von Containern auf Rechenknoten eines Rechenclusters erfolgte automatisiert in einem einzelnen Job-Script (sbatch im Fall von CARO/CARA), das mit mehreren `SRUN` Kommandos zunächst den ETCD server, dann die Meta- und Storageserver des verwendeten Speichersystems (IML oder BeeGFS) und zu guter Letzt die Anwendung (etwa eine Simulation mit FSDM und CODA) mittels `apptainer run` Kommandos in ihren jeweiligen Containern startet.

Erste Performanzmessungen unter Verwendung dieses Deployments zeigten deutlich geringere Werte als erwartet. Die weitere Analyse dieser Abweichung ergab, dass im beschriebenen Setup zwei Speichersystemen hintereinander geschaltet waren. Dies ergibt sich dadurch, dass in dieser Konfiguration alle Rechenknoten zur Datenablage das clustereigene Lustre Dateisystem verwendeten und BeeGFS und IML Dateisysteme ihre persistenten Daten in das Lustre Dateisystem schrieben. So wurden alle Daten mehrfach über das Netzwerk übertragen und teilweise neu gestriped, was zu zusätzlichem Overhead führte.

Um diesen systembedingten Flaschenhals zu reduzieren, wurde im weiteren Verlauf ein leicht modifiziertes Setup gesetzt: CARA und CARO bieten für einen (relativ kleinen) Anteil ihrer Rechenknoten knotenlokale NVMes. Auf diesen $1\,\mathrm{TB}$ großen NVMe SSDs, die für Jobs an einem lokalen Mountpoint im Unix-Dateisystem zur Verfügung stehen, wurden die Daten von den Backends in weiteren Tests direkt abgelegt. Dieses Setup ist in Abbildung 12 veranschaulicht.

Mit diesem optimierten Deployment konnten realistischere Performanzwerte erzielt werden. Die durchgeführten Deployments demonstrieren, dass die entwickelten Komponenten flexibel, ohne administrative Eingriffe und unter realen HPC-Bedingungen betrieben werden können. Damit wurde in AP6.b die praktische Einsetzbarkeit der entwickelten Architektur erfolgreich nachgewiesen.

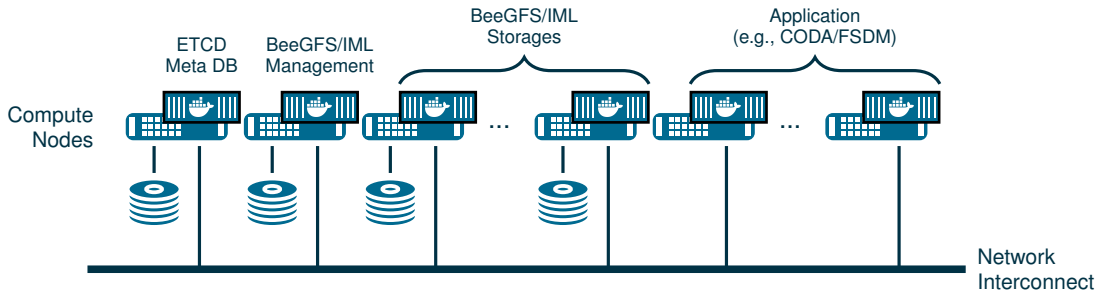

Abbildung 12: Deployment Setup auf Cluster Knoten mit knoten-lokalen NVMes

AP6.c Benchmarking und Tuning (Leitung: GAUG/U) Im Rahmen von AP6.c wurden umfassende Benchmark- und Tuning-Maßnahmen durchgeführt. Als wichtigstes initiales Leistungskriterium wurde neben der Korrektheit der ausgeführten Speicheroperationen der Speicherdurchsatz von Lese- und Schreiboperationen identifiziert. Hierbei wurde bewusst ein vollständiger I/O-Pfad von der Applikation durch libIOVerbs bis in die Backends und den unterliegenden Speicher betrachtet. Neben der Backend-Implementierung selbst beeinflussten insbesondere die Ausführungsumgebung (Maschinenanzahl, Art des unterliegenden Speichers, Parallelität), als auch die zusätzlichen libIOVerbs Komponenten wie Metadatenkomponente die gemessene Leistung.

Als repräsentative Ende-zu-Ende-Anwendung wurde FSDM eingesetzt, da diese in hohem Maße vom Lese- und Schreibdurchsatz profitiert und damit geeignet ist, diesen Aspekt der Leistungsfähigkeit des Systems realitätsnah zu bewerten.

Der Speicherdurchsatz wurde für mehrere verschiedene Deployments vom IML und BeeGFS gemessen. Dabei wurden neben den Performanz Werten auch Skalierungsgrenzen der Backends auf das vergrößerte Deployment identifiziert und gezielt adressiert. Insbesondere zeigte der BeeGFS User-Space Client dabei Synchronisationsprobleme bei Schreibvorgängen mit hoher Thread-Parallelität, die in Zusammenarbeit zwischen GAUG/U und DLR untersucht und behoben werden konnten. Die Verbesserungen betrafen hier sowohl den BeeGFS Client selbst, als auch kleinere Fehler im MPI basierten Test. Die Benchmarking-Phase diente somit nicht nur der Leistungsbewertung, sondern auch als systematischer Feedback-Mechanismus zur Verbesserung der Implementierung auf mehreren Ebenen.

Die durchgeführten Tuning-Maßnahmen führten zu einer stabileren Skalierung und verbesserten Parallelitätseigenschaften im Backend. Damit konnte im AP6.c die Funktions- und Leistungsfähigkeit des vollständigen I/O-Stacks unter realistischen Bedingungen evaluiert und identifizierte Skalierungsprobleme adressiert werden.

AP6.d Auswertung (Leitung: GAUG/U) Im Rahmen von AP6.d wurde auf dem DLR-Cluster CARO das in Abschnitt AP6.b beschriebene Deployment verwendet, um Ende-zu-Ende Messungen von IOVerbs für den CFD Anwendungsfall inklusive der im Rahmen von AP1.d entworfenen High-Level API für Strömungsmechanik durchzuführen und zu evaluieren. Hierzu wurde erneut das im Abschnitt zu AP5.e beschriebene Netz vom 4. High-Lift-Prediction Workshop in Verfeinerungsstufe E verwendet.

Aufgrund der begrenzten Verfügbarkeit von Knoten mit knotenlokalen NVMes auf CARO und Einschränkungen im Schedulingsystem, die einen gleichzeitigen Einsatz dieser Knoten mit anderen Knoten ohne lokalen SSD-Speicher nicht erlauben, mussten auch Knoten, auf denen letztlich die Applikation läuft, aus diesem beschränkten Pool allokiert werden. Um Messungen dennoch in vertretbarer Zeit durchführen zu können, wurden pro Job zehn NVMe-Knoten allokiert, von denen auf zweien der ETCD Metadatenserver bzw. der storagebackend-spezifische Managementserver ausgeführt wurde, auf vieren die BeeGFS oder IML Storages, und auf den verbleibenden vieren die eigentliche Applikation. Um einen möglichst fairen Vergleich zu ermöglichen, wurden für Referenzmessungen mit MPI-IO auf CAROs Lustre-Dateisystem daher ebenfalls vier SSD-basierte Object Storage Targets (OSTs) verwendet.

Insgesamt wurden so jeweils fünf Messungen für lesende und schreibende Zugriffe durchgeführt:

- MPI-IO in 4 Lustre SSD OSTs
- IOVerbs in 4 IML Storages mit 4 Lustre SSD OSTs als Speicher

- IOVerbs in 4 IML Storages mit knoten-lokalem Speicher
- IOVerbs in 4 BeeGFS Storages mit 4 Lustre SSD OSTs als Speicher
- IOVerbs in 4 BeeGFS Storages mit knoten-lokalem Speicher

Abbildung 13 zeigt die gemessenen Laufzeiten für Lesen (links) und Schreiben (rechts) der initialen IOVerbs implementierung. Mit dem aktuellen Stand der Implementierung bleibt die Performanz von IOVerbs sowohl mit IML als auch BeeGFS als Backend hinter der von MPI-IO zurück. Dies kann auf mehrere Faktoren zurückgeführt werden. Zunächst muss festgehalten werden, dass bis Projektende nur eine erste Version der IOVerbs Bibliothek fertiggestellt und hier vermessen werden konnte, sodass davon auszugehen ist, dass noch einige Optimierungen an deren Implementierung möglich sind, um ihre Performanz weiter zu steigern. Auf der anderen Seite ist MPI-IO eine über Dekaden optimierte Bibliothek, sodass eine vergleichbare Perfomanz zum aktuellen Projektstand nicht realistisch zu erwarten war. Darüber hinaus ist auch die konkrete MPI-IO Installation auf CARO für optimale Performanz durch langfristige Tests mit dem verwendeten Lustre Dateisystem von den Betreibern eingestellt worden und es wurden “best-practices” für die Striping-Parameter von Lustre identifiziert. Eine entsprechend umfangreiche Parameterstudie zur Optimierung solcher Parameter in IML oder BeeGFS konnte in Rahmen des Projektes nicht geleistet werden, sodass mit einer rudimentären Parametrisierung der Dateisysteme gemessen wurde.

Speziell für IML fällt auf, dass eine Verwendung oberhalb des Lustre Dateisystem zu deutlich erhöhten Laufzeiten (ca. 5-8x länger als MPI-IO) führte. Bei Verwendung knotenlokaler NVMes (was den faireren Vergleich mit MPI-IO darstellt) konnte die Geschwindigkeit jedoch deutlich verbessert werden (ca. 2-3x langsamer als MPI-IO). Dies unterstreicht den Einfluss der doppelten Speicherung über Lustre im ursprünglichen Setup.

Vergleicht man BeeGFS und IML, so fällt auf, dass der BeeGFS Client auf dem Lustre Backend besseren Durchsatz zeigt. Die Architektur von BeeGFS in der auf der Storage Server Seite je nach Clientanforderung Schreib- und Lesezugriffe gecacht werden können, sorgt hier für eine größere Unabhängigkeit der Systemperformanz vom Durchsatz auf das jeweilige Storage Medium.

Auf der anderen Seite zeigt BeeGFS praktisch keine Perfomanzunterschiede zwischen der Verwendung des verteilten Lustre Dateisystems und Knoten-lokaler NVMes. Insgesamt bleibt die Geschwindigkeit in beiden Fällen circa um Faktor 3 (schreiben) bis 6 (lesen) hinter der von MPI-IO zurück. Der fehlende Unterschied zwischen den unterschiedlich schnellen Backends deutet darauf hin, dass das Performanz Bottleneck nicht in der Storage Schicht selbst, sondern auf der Client Seite oder im Netzwerk zwischen Client und Storage liegt. Hierbei ist wichtig zu bemerken, dass die vorliegenden Zahlen mit einer für BeeGFS relativ kleinen Chunkgröße von 64KiB gemessen wurden. MPI-IO verwendet beim direkten Schreiben deutlich größere Chunks (2MiB), was zu einer Verringerung der nötigen Netzwerktransaktionen führt. Es ist wahrscheinlich, dass der BeeGFS Client bei einer Wahl größerer Chunks bessere Performanz zeigen würde. Zusätzlich zur manuellen Konfiguration der globalen Default Chunkgröße liegt für den BeeGFS ein Patch vor, der die Chunkgröße basierend auf den Anforderungen der Applikation pro Collection festlegen kann. Dieser Patch konnte im Projektzeitraum jedoch leider nicht mehr im größeren Maßstab getestet werden. Eine weitere potentielle Quelle für zusätzliche Latenz und verringerten Durchsatz ist die komplexere Synchronisierung im zweistufigen Mechanismus für asynchronen I/O zwischen dem C++ Backend für libIOVerbs und der asynchronen Rust I/O Schicht im `beegfs_backend`. Hier sind weitere Messungen und Analysen notwendig, um das bestehende Bottleneck einzugrenzen und eventuelle Verbesserungen zu implementieren. Abschließende Aussagen über die Relevanz des unterliegenden Storage Systems auf Server Seite lassen sich erst treffen, wenn im I/O Pfad davor liegende Bottlenecks verstanden und behoben sind.

Zusammenfassend zeigen die Messungen, dass die Ende-zu-Ende-Funktionalität des IOVerbs-Stacks im CFD-Anwendungsszenario erfolgreich demonstriert werden konnte. Die Performanz liegt erwartungsgemäß unterhalb der eines langjährig optimierten Referenzsystems, jedoch wurden konkrete Optimierungsansätze identifiziert. Damit ist eine belastbare Grundlage für weiterführende Performanzoptimierungen in zukünftigen Projekten geschaffen. Da eine Ende-zu-Ende Einsatzfähigkeit der libIOVerbs und der Backendkomponenten gezeigt werden konnte, steht einem experimentellen Einsatz der entwickelten Produkte in Pre-Exascale und Exascale Systemen nichts im Weg. Ein solcher Einsatz könnte in Verbindung mit einem Austausch mit den Projektpartnern zu einer Erweiterung der getesteten Einsatzszenarien und damit zu zusätzlichen Informationen, die in die Optimierung der durch libIOVerbs und die Backends unterstützten Operationen beitragen.

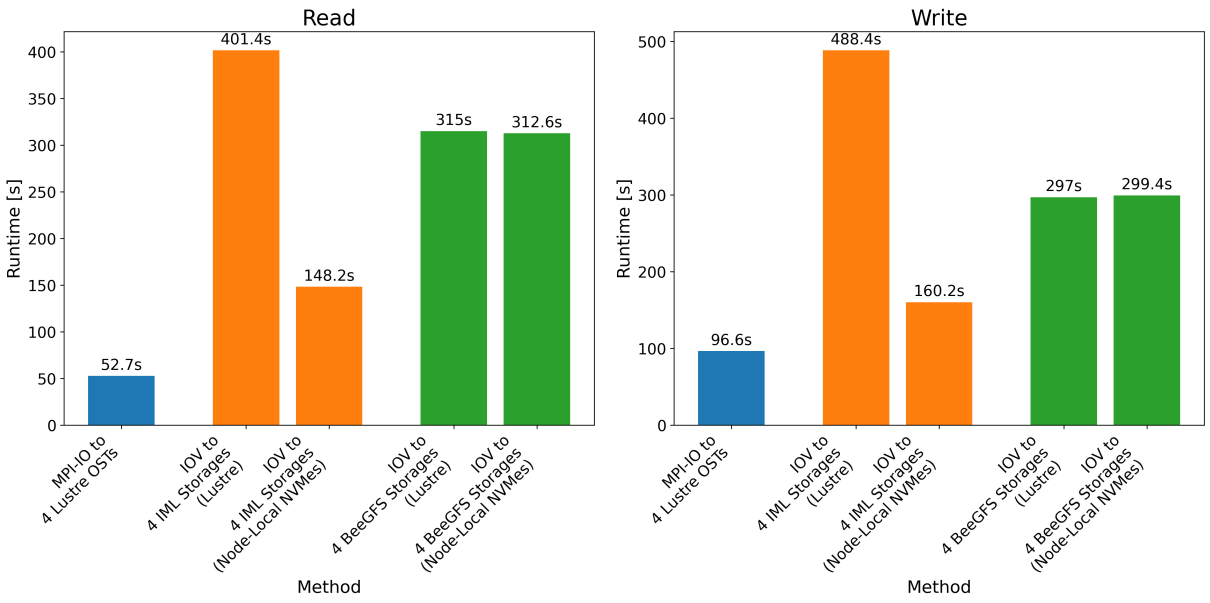

Abbildung 13: Benchmark Ergebnisse für CFD Usecase

2.1.7 AP7: AP7: Management und Qualitätssicherung (Leitung: GAUG)

AP7.a Projektleitung (Leitung: GAUG) Im Rahmen der Projektleitung wurde zunächst ein Projekthandbuch angelegt und verteilt. Des weiteren wurden die jährlichen Zwischenberichte und mindestens ein jährliches Vor-Ort-Treffen zwischen den Projektpartnern organisiert. Die vom Projekt erzielten Ergebnisse wurden jährlich zur Präsentation auf verschiedenen Konferenzen, wie z.B. der ISC Konferenz in Hamburg oder der HPC-Statuskonferenz, zur Dissemination präsentiert. Zur Koordination der Projektpartner fand ein monatliches Online-Treffen und spezifisch für die Definition der IOVerbs API für den größten Teil des Projekts existierte außerdem ein wöchentliches Meeting zwischen den an der API-Entwicklung beteiligten Projektpartnern.

AP7.b Qualitätssicherung (Leitung: GAUG) Zur Qualitätssicherung wurden CI/CD-Pipelines zum automatisierten Testen der entwickelten Software implementiert. Des weitere wurden Coding-Standards zur Vereinheitlichung des Quelltextes definiert.

AP7.c Risikomanagement (Leitung: GAUG) Durch die verspätete Einstellung von Personal für das MCSE-Projekt und die benötigte Einarbeitung ergab sich ein allgemeiner Projektrückstand. Aufgrund dessen haben wir eine Verschiebung des Hauptmeilensteins um 6 Monate beantragt und bewilligt bekommen. Alle Projektpartner waren bestrebt, den Rückstand durch fokussierte Anstrengungen und enge Zusammenarbeit aufzuholen. Dies ist vollkommen gelungen und ergab einen erfolgreichen Abschluss des Hauptmeilensteins im September 2024. Zur weiteren Verbesserung der erzielten Ergebnisse wurde das Projekt nach dem ursprünglichen Projektende um weitere 3 Monate kostenneutral verlängert.

2.2 Wichtigste Positionen des zahlenmäßigen Nachweises

Nr.	Position	Zuwendung	IST	Aktuell verbleibende Mittel	Bemerkungen
0812	Beschäftigte E12-E15	216.713,00 €	216.230,80 €	482,20 €	Die verausgabten Mittel entsprechen in Höhe und Position des Gesamtfinanzierungsplans der Planung. Die Abweichung in Position 0846 wurde über die anderen freien Mittel abgefangen.
0822	Beschäftigungsentgelte	27.918,00 €	26.848,52 €	1069,48 €	
0835	Vergabe von Aufträgen	340.800,00 €	340.663,67 €	136,33 €	
0846	Dienstreisen	8615,00 €	10.184,85 €	-1.569,85 €	
0861	**Gesamtausgaben**	**594.046,00 €**	**593.927,84 €**	**118,16 €**	

2.3 Notwendigkeit und Angemessenheit der geleisteten Arbeit

Das Projektkonsortium gewährleistet durch seine Zusammensetzung aus Universitäten, Forschungseinrichtungen, die HPC-Systeme betreiben, Nutzenden von HPC-Software und dem KMU eine interdisziplinäre Bearbeitung des Forschungsthemas, welches aufgrund der hohen Komplexität der Aufgabenstellung und dem wirtschaftlichen Risiko nur mit Fördermitteln finanzierbar ist. Bei der Entwicklung einer neuen I/O-Schnittstelle sind Umsetzung und Nachhaltigkeit hohe Risiken, die nur in einem Konsortium bewältigt werden können. Das IML System von Fraunhofer bietet die Möglichkeit eine User-Space Lösung umzusetzen und gleichzeitig mit den IOVerbs und MCS2 ein neues I/O-System für Supercomputer in den Einsatz zu bringen. Die Entwicklung einer skalierbaren, performanten Metadateninfrastruktur und Ausfallsicherheitsmechanismen hierfür benötigen viel Zeit und sind mit wissenschaftlichen und hohen wirtschaftlichen Risiken verbunden. Für das DLR ist die numerische Simulation eine entscheidende Schlüsseltechnologie für die vollständige digitale Beschreibung von Luftfahrzeugen. Die numerische Strömungssimulation nimmt in den multidisziplinären Prozessketten eine zentrale Rolle ein. Die effiziente Nutzung aktueller und künftiger Hochleistungsrechencluster ist daher von entscheidender Bedeutung. Mit der anstehenden Verfügbarkeit von Exascale-Rechnern wird die hocheffiziente E/A eine wichtige Rolle spielen. Nur durch die Durchführung der geplanten Entwicklungen kann ein revolutionärer Ansatz, der zu höherer Skalierbarkeit führt, umgesetzt werden. Eine DLR-interne Förderung für den Schwerpunkt E/A-Skalierbarkeit für Hochleistungsrechner der Exascale-Klasse ist aktuell nicht möglich. Die GAUG und TUD hätte zwar die Möglichkeit auch ohne Förderung durch das BMBF das Projekts in kleinen Arbeiten aufzuteilen und in Studentenarbeiten bearbeiten zu lassen, allerdings nicht auf eine konsistente Weise und in dem Umfang der für eine erfolgreiche Umsetzung notwendig ist. Zudem erfordert eine nachhaltige Lösung einen erheblichen Entwicklungsaufwand in Bezug auf die Umsetzung und Demonstration der Effizienz und Leistungsfähigkeit. Die geplanten Vorhaben sind deshalb nur im Verbund mehrere Partner als Projekt zu bewerkstelligen und können nicht von einem einzelnen Partner umgesetzt werden.

Das Projekt stellt auch für den Industriepartner ein großes Risiko da, da nicht garantiert werden kann, dass die Schnittstelle sofort auf breite Akzeptanz bei relevanten Anwendungen finden wird.

Zusammengefasst sind relevante Risiken vor allem technischer Natur und betrifft insbesondere die folgenden Komponenten: IO-Verbs API, der Weiterentwicklung von BeeGFS und IML zu MCSE, dem Workflow System für Kampagnen-Speicher und der Portierung von FSDM für direkten Speicherzugriff. Die Risiken hinsichtlich der einzelnen Komponenten sind wie folgt.

IO-Verbs API Bei der Entwicklung einer API ist ein Risiko, dass die Schnittstelle nicht auf Akzeptanz bei relevanten Anwendungen und der HPC-Gemeinschaft finden wird. So besteht die Gefahr, dass beim Schnittstellendesign wichtige Nutzungsmuster und Anwendungsanforderungen übersehen werden könnten, die jedoch wiederum für eine breite Akzeptanz der Schnittstelle Voraussetzung sind.

Weiterentwicklung von BeeGFS und IML zu MCSE Ein elementares Risiko ist die Komplexität dieser Pakete, wenngleich Entwickler im Konsortium beteiligt sind, besteht die Gefahr, dass die Integrationsarbeit deutlich unterschätzt wird oder dass die entwickelte IO-Verbs API nicht auf die verfügbare Architektur der bestehenden Softwarepakete eignet.

Workflow System für Kampagnen-Speicher Risiken bei der Entwicklung des Workflows sind zweierlei. 1) Zu Validierungszwecken der Forschungsergebnisse muss die entwickelte Workflow-Lösung in einem hinreichend großen Testsystem (oder dem Produktivsystem) deployed werden können und insbesondere für mehre Tage bis Wochen kontinuierlich ausgeführt werden – dies führt zu technischen Herausforderungen im Betrieb und Anforderungen an die technische Reife der Lösung. 2) Das Gesamtsystem basiert auf mehreren, in sich bereits komplexe Softwarekomponenten; die Integration ist daher nicht trivial und erfordert umfassendes Wissen der beteiligten Entwickler, Updatezyklen der Software bspw. aufgrund von notwendigen Sicherheitsupdates können dazu führen, dass während des Projektverlaufs die neu entwickelte Software mehrfach anpasst werden muss, was zusätzlich Zeit erfordert.

Portierung von FSDM für direkten Speicherzugriff Beim direkten Speicherzugriff muss der E/A-Pfad der Anwendung invasiv überarbeitet werden, es besteht das Risiko hierbei ungewünschte Seiteneffekte (Bugs) einzuführen und notwendige Änderungen (aufgrund der Code Größe und Komplexität) zu übersehen. Schließlich ist die langfristige Integration in die Code-Basis ungewiss, da diese Abstimmungen mit allen Entwicklern erfordert.

Sollten sich derlei Risiken manifestieren (Schwierigkeiten ergeben), wird dies gemäß Arbeitsplan in den Meilensteinen bspw. dem zentralen Meilenstein festgestellt und entsprechend gehandhabt.

Die Förderprogramme der EU[33], des BMBF und der DFG zeigen aktuell keine alternative Fördermöglichkeit für das MCSE Projekt auf.

2.4 Voraussichtlicher Nutzen, insbesondere Verwertbarkeit des Ergebnisses im Sinne des fortgeschriebenen Verwertungsplans

2.4.1 Wirtschaftliche Erfolgsaussichten

Künftige HPC-Systeme - insbesondere solche der Exascale-Klasse mit heterogenen und hierarchischen Speichertechnologien - profitieren von einem schnellen und flexiblen E/A-Stack und bilden einen wirtschaftlich relevanten Markt[34]. Bestehende Lösungen bauen auf Low-Level Standardschnittstellen wie POSIX oder S3, oder auf High-Level Schnittstellen wie HDF5 oder ADIOS. Dazwischen ist eine große semantische Lücke, welche durch die IOVerbs ausgefüllt werden könnte. Die Definition und Einführung der IOVerbs Schnittstelle ermöglicht die Beschreibung der Parallelität von Anwendungen und damit das effiziente Betreiben verteilter Speichersysteme. Auf der einen Seite können Anwender:innen und Entwickler:innen domänenspezifische High-Level Bibliotheken entwickeln und die nötige Semantik und Konsistenzanforderungen, sowie die verwendeten E/A Zugriffsmuster, die sie benötigen, präzise mit den IOVerbs ausdrücken. Somit ist es möglich flexibel auf die Anforderungen zu reagieren, z.B., eventuelle Caches zuzuschalten bzw. komplexe Synchronisationsmechanismen abzuschalten, um bestimmte Konsistenzanforderungen zu garantieren. Auf der anderen Seite werden Dateisystementwickler:innen durch die IOVerbs in die Lage versetzt einen gegebenen E/A Workload optimal zu implementieren. D.h. die Resultate aus diesem Projekt tragen langfristig dazu bei die Entwicklung neuer Speichersysteme, als auch Anwendungen, signifikant zu verbessern.

Als Alternative zur klassischen POSIX-I/O-Schnittstelle im Linux VFS ist es ein Ziel des Projektes, eine IOVerbs-Schnittstelle direkt im Linux-Kernel zu integrieren und damit einen neuen E/A-Standard zu schaffen. Hierfür werden wir bei Erfolgsaussicht noch während der Projektlaufzeit (spätestens Jahr 3), unmittelbar weitere Aktivitäten starten.

Kommerzielle Verwertung: Mit der Entwicklung des MCS2 wird aus dem IML System von Fraunhofer ein neues Memory Centric Storage System auf TRL-6, welches die Möglichkeiten einer weitergehenden kommerziellen Verwertung durch einen Industriepartner ermöglicht. Das ITWM selbst wird das System in

[33] EU: `https://europa.eu`

[34] Ca. $41 Milliarden Volumen in 2022:
`https://www.hpcwire.com/off-the-wire/total-hpc-market-revenue-grew-5-2-to-41-0-billion-in-2021-says-intersect360-research/`

Projekten mit Kunden, zum Beispiel zur Lösung von I/O-Problemen bei KI-Workflows, in der Folge kommerziell einsetzen. Des weiteren bietet sich die Einbindung in dem auf GPI-Space basierenden ALOMA Framework an, dass eine Low-Code Programmierumgebung zur Erstellung komplexer Processing Workflows anbietet.

Die Weiterentwicklungen für das BeeGFS Dateisystem werden durch den Industriepartner ThinkParQ mittelfristig weiterentwickelt und in den Vertrieb gebracht. Die entwickelte Alpha Version des BeeGFS User-Space Clients wird auf die Überführung in eine Produktentwicklung untersucht. Ein Zeithorizont ist zum jetzigen Zeitpunkt schwer abzuschätzen, wird aber zusätzlich mindestens 18-24 Personenmonate Entwicklungszeit benötigen. Insbesondere für Hoch- und Höchstleistungsrechenzentren wäre somit eine interessante Alternative zu den klassischen Ansätzen von parallelen Dateisystemen gegeben, was letztendlich einer höheren Performance des Gesamtsystems bei gleichzeitig besserer Nutzung aller zur Verfügung stehenden Ressourcen des Storage-Backends zur Folge hätte. Mit der Umsetzung eines User-Space Clients und der Implementierung der IOVerbs-Schnittstelle wird ein Vorsprung vor Konkurrenzprodukten, wie zum Beispiel Lustre, das derzeit einen hohen Marktanteil in Höchstleistungsrechenzentren hat, erarbeitet, womit Marktanteile für BeeGFS gewonnen werden können. Das Marktmodell der Firma ThinkParQ besteht darin, Supportverträge und Lizenzen für BeeGFS zu vertreiben sowie Support bereitzustellen. Damit profitiert ThinkParQ direkt von einer höheren Anzahl an Neuinstallationen.

2.4.2 Wissenschaftliche und/oder technische Erfolgsaussichten

MCSE hat bei erfolgreicher Umsetzung eine große wissenschaftlich-technische Reichweite. Neben den Berichten, die im Verlauf des Projektes erstellt werden, sind Veröffentlichungen in Fachzeitschriften und Konferenzen geplant, die in enger Abstimmung mit den beteiligten Projektpartnern bereits während der Projektlaufzeit umgesetzt werden sollen. Die Ergebnisse und Erkenntnisse werden von der GAUG in Lehrveranstaltungen integriert, bspw. die Vorlesung “High-Performance Data Analytics”, das Seminar “Newest Trends in High-Performance Data Analytics” und der “Practical Course in High-Performance Computing”. Außerdem wurden die letzten Jahre zwei einwöchige Summer Schools für die Klima- und Wettergemeinschaft organisiert, welche auch Training für I/O anboten - diese Trainings sollen ausgebaut werden. Analog zu IBVerbs bieten IOVerbs eine Front für weitere Arbeiten und Forschungsmöglichkeiten. Zu den Themenbereiche sollen vielfältige BSc-, MSc- und PhD-Arbeiten angeboten werden.

Die Erkenntnisse und Ergebnisse werden im Center of Excellence in Simulation of Weather and Climate in Europe (ESiWACE), an dem Prof. Kunkel beteiligt ist, einfließen. Die Erweiterungen und Verbesserungen für ESDM können von der wissenschaftlichen Gemeinschaft direkt genutzt und bspw. im ESiWACE Projekt für dessen Anwendungen evaluiert werden. Weiterhin profitieren die IO500, die “IO500 Foundation” und die Standard-Benchmarks MDTest und IOR ebenfalls von dem Projekt. Prof. Kunkel plant auch weitere Projekte zur effizienten E/A, welche selbstverständlich die Ergebnisse dieses Projekt berücksichtigen werden. Insgesamt wird durch das MCSE Projekt die Standort-Kompetenz aller Partner gefördert und für weitere Aktivitäten attraktiver.

Die GAUG bildet zusammen mit der GWDG den Tier 2 Standort im Hochleistungsrechnen NHR@Göttingen. Der Standort verfügt über ca. 2000 Nutzende. Das Campus-Institute für Data Science (CIDAS) der Universität hat großes Interesse daran datengetriebene Workflows effizient abarbeiten zu können. Die IOVerbs API hat das Potential die E/A-Leistung für verschiedenste Workflows zu verbessern und somit den Wissenschaftsoutput zu steigern. Die GWDG[35] hat starkes Interesse daran die Ergebnisse der GAUG im operationalen Bereich zu verwenden und weiter zu verbessern.

Das DLR erwartet durch die Projektergebnisse eine Stärkung der bereits in mehreren Luftfahrtunternehmen produktiv genutzten Simulationsplattform FlowSimulator Data Manager (FSDM), sowie des Strömungslösers CODA. Neben CODA profitieren alle HPC-Anwendungen, die im FlowSimulator eingebunden sind, von der Projektergebnissen, weil die IML-/IOVerbs-Erweiterungen in der E/A-Schicht des FSDM Frameworks umgesetzt werden. Dies umfasst u.a. die Strömungslöser TAU und TRACE sowie Strukturlöser, Netzadaption oder -deformation. Erweiterungen und Verbesserungen von DLR-Software steigern deren Attraktivität weiter und verbessern die Stellung des DLR in der Forschungs- und Industrielandschaft.

Technisch birgt das Projekt für den Industriepartner auf vielfältige Weise ein großes Potential: Aktuell wird im BeeGFS-Kernel-Client viel Prozessorzeit durch das Kopieren von Daten zwischen User-Space und Kernel-Space gebunden. Eine Implementierung des BeeGFS Clients im User-Space hat somit eine signifi-

[35] GWDG:https://www.gwdg.de

kante Verringerung der CPU Nutzung und des Kopierens von Daten im Arbeitsspeicher zur Folge, was zu einer niedrigeren Latenz und einem höheren Durchsatz im Datenpfad führt. Außerdem werden zusätzliche Prozessorzyklen, die sonst für E/A verwendet werden müssten, auf den Rechenknoten für Berechnungen frei was die Rechenknoten ihre Hauptaufgabe effizienter erfüllen lässt und zu einer Erhöhung der Rechenleistung führt. Durch die Open Source Entwicklung von IOVerbs wird eine Schnittstelle geschaffen, an die Anwendungen einfach angebunden werden können und die dadurch schnell eine breite Unterstützung finden kann.

An der TU Dresden werden die Ergebnisse und Erkenntnisse des Projekts in verschiedenen Bereichen genutzt. Den zentralen universitären Aufgaben der Erforschung, Entwicklung und Erarbeitung neuer wissenschaftlicher Erkenntnisse und Methoden entsprechend sowie deren Integration in die Hochschulausbildung und Qualifikation des akademischen Nachwuchses, sollen die IOVerbs in die Forschung und Lehre integriert werden. Das Thema bietet großes Potenzial für Forschungsarbeiten wie BSc-, MSc- sowie PhD-Arbeiten, die an der TU Dresden bearbeitet und betreut werden können. Zudem ermöglichen es die IOVerbs, die E/A-Anforderungen paralleler Anwendungen präzise zu beschreiben und damit performanter für ein gegebenes System abzubilden. Als eines von neun NHR-Zentren in Deutschland liegt ein Schwerpunkt des ZIH auf Datenanalyse und Speichersystemen; es besteht daher ein besonderer Bedarf an einer optimalen Auslastung der Speichersysteme. Die IOVerbs API hat das Potenzial, diese Anforderungen zu erfüllen und damit die Produktivität datenintensiver Anwendungen zu verbessern.

2.4.3 Wissenschaftliche und wirtschaftliche Anschlussfähigkeit

Übergreifend: Bei Erfolg sind die IOVerbs ein Durchbruch für die Nutzung von E/A-Systemen insbesondere in HPC, aber auch darüber hinaus und bieten ein außerordentliches Potential für die Wissenschaft, als auch für die Wirtschaft. Analog zur Revolution für Datenbankanfragen mittels SQL können die semantischen Anforderungen von Anwendungen spezifiziert und von Systemen optimiert werden. Es ist jedoch nicht zu erwarten, dass in einem Projekt die IOVerbs finalisiert werden könnten; analog zu SQL wird eine Weiterentwicklung unumgänglich sein. Eine Patentierung der IOVerbs ist nicht vorgesehen, da dies Konkurrenzaktivitäten und Replikation fördern und somit den Nutzen für die HPC-Gemeinschaft stark einschränken würde. Wir planen, die Spezifikation und Entwicklung der IOVerbs offen als Open Source Lösung anzubieten. Es soll spätestens gegen Ende der Projektlaufzeit, aber so frühzeitig wie möglich, ein möglichst internationales Konsortium etabliert werden, welches die Kuration der IOVerbs langfristig übernimmt - gleichzeitig verbleibt den deutschen Projektpartnern eine zentrale Rolle in diesem Konsortium. Bei Erfolgsaussicht werden wir im Nov. 2024 auf der Supercomputing eine entsprechende Birds-of-a-feather Session (BoF) anbieten oder die IOVerbs als Bestandteil der “Analyzing Parallel I/O” und “IO500” BoF vorstellen. Die Modifikationen der IO500 dienen als Multiplikator, denn die IO500 BoFs auf ISC und Supercomputing sind gut besucht und auch für die internationale Presse attraktiv. Wir werden versuchen die Modifikationen im Rahmen der BoFs zu platzieren.

Das Fraunhofer ITWM plant das im Rahmen dieses Projektes entwickelte MCS2 Software System in Kundenprojekten weiter zu entwickeln, um Produktreife zu erlangen und schließlich durch ein zu gründendes Spin-Off oder einen vorhandenen Industriepartner, wie die ThinkParQ GmbH, weiter zu vermarkten. Die GAUG plant den potentiellen Nutzern und Industriepartnern Beratung anzubieten und bei der kommerziellen Verwertung von MCS2 (beim ITWM) mitzuwirken.

Der Industriepartner zielt auf die Weiterentwicklung des BeeGFS User-Space Clients und der Integration der IOVerbs Schnittstelle zum Produktstatus ab. Durch eine Integration verschiedener Speichertechniken in den E/A-Workflow von Anwendungen würden zusätzliche Programme zur Klassifizierung und zum Verschieben der verwendeten Daten nötig. Die Entwicklung solcher Programme könnte in einem Anschlussprojekt realisiert werden während parallel dazu die Weiterentwicklung der Projektergebnisse zum Produkt durchgeführt würde. Dabei könnte auch untersucht werden, inwieweit sich künstliche Intelligenz zur Datenklassifizierung einsetzen lässt.

2.5 Während der Durchführung des Vorhabens dem ZE bekannt gewordener Fortschritt auf dem Gebiet des Vorhabens bei anderen Stellen

Während der Durchführung des Vorhabens wurden im Kerngebiet des Antrags von Low-Level-Storage-APIs keine neuen Entwicklungen dem ZE bekannt.

2.6 Erfolgte oder geplante Veröffentlichungen von Ergebnissen

2.6.1 Poster

Der Projektfortschritt wurde jeweils im Rahmen von drei ISC-Konferenzen als Poster präsentiert.

ISC 2024 Projekt-Poster Vietinghoff et al. (2024)

ISC 2025 Projekt-Poster Vietinghoff et al. (2025)

ISC 2026 Projekt-Poster Vietinghoff et al. (2026)

2.6.2 Vorträge

HPC Statuskonferenz 2023 Vortrag mit anderen SCALEXA Projekten Laure et al. (2023)

ISC 2024 BOF über IO500 Mockups Kunkel (2024)

OpenFOAM Workshop 2024 Vortrag über Konzepte, um IOVerbs in OpenFOAM zu integrieren Höhn (2024)

HPC Statuskonferenz 2024 Vortrag mit anderen SCALEXA Projekten Laure et al. (2024)

Monthly Storage Talks 2024 Vortrag über Storage Semantics

Monthly Storage Talks 2025 Vortrag über die seinerzeitigen Ergebnisse des MCSE Projekts

HPC Statuskonferenz 2025 Vortrag mit anderen SCALEXA Projekten Laure et al. (2025)

2.6.3 Seminare und Workshops

HLRS 2025 Präsentation mit Workshop zu MCS2 für das HLRS in Stuttgart

SURF 2025 Präsentation mit Workshop zu MCS2 für Surf in Amsterdam

EOFS 2025 Präsentation mit Workshop zu MCS2 für: "3rd EOFS/NHR Workshop on Open Parallel File Systems".

2.6.4 Exponate und Demonstrationen

SC 2024 Alleinstehendes Exponat mit IML Live Demonstration

SC 2025 Alleinstehendes Exponat mit IML Live Demonstration

ISC 2025 Alleinstehendes Exponat mit IML Live Demonstration

SC 2026 Alleinstehendes Exponat mit IML Live Demonstration

ISC 2026 Alleinstehendes Exponat mit IML Live Demonstration

2.6.5 Publikationen

Frontiers Semantik Artikel Oeste et al. (2025)

Literatur

Bingert, S., Köhler, C., Nolte, H., and Alamgir, W. (2021). An api to include hpc resources in workflow systems. In *INFOCOMP 2021, The Eleventh International Conference on Advanced Communications and Computation*, pages 15–20.

Decker, J. (2025). Achieving scalable ai inference in a unified cloud and hpc environment by employing system of systems architecture design.

Höhn, P. (2024). Proposal for a new storage backend in OpenFOAM. Number 19 in The 19th OpenFOAM Workshop (OFW19). OpenFOAM Workshop.

Jette, M. A. and Wickberg, T. (2023). Architecture of the slurm workload manager. In Klusáček, D., Corbalán, J., and Rodrigo, G. P., editors, *Job Scheduling Strategies for Parallel Processing*, pages 3–23, Cham. Springer Nature Switzerland.

Kunkel, J. (2024). Vi4io/io500-ior-mockup: An io500 mockup to showcase potential of new interfaces latest.

Kunkel, J., Lofstead, J., and Bent, J. (2017). The Virtual Institute for I/O and the IO-500.

Laure, E., Höhn, P., and Dwivedi, A. K. (2025). Speichersysteme für das Exascale. HPC-Statuskonferenz 2025.

Laure, E., Kunkel, J., and Vogel, P. (2023). Speichersysteme für das Exascale. HPC-Statuskonferenz 2023.

Laure, E., Kunkel, J., and Vogel, P. (2024). Speichersysteme für das Exascale. HPC-Statuskonferenz 2024.

Mölder, F., Jablonski, K., Letcher, B., Hall, M., Tomkins-Tinch, C., Sochat, V., Forster, J., Lee, S., Twardziok, S., Kanitz, A., Wilm, A., Holtgrewe, M., Rahmann, S., Nahnsen, S., and Köster, J. (2021). Sustainable data analysis with snakemake [version 2; peer review: 2 approved]. *F1000Research*, 10(33).

Oeste, S., Höhn, P., Kluge, M., and Kunkel, J. (2025). An analysis of the i/o semantic gaps of hpc storage stacks. *Frontiers in High Performance Computing*, Volume 3 - 2025.

The HDF Group (1997–2026). Hierarchical Data Format, version 5.

Vangoor, B. K. R., Tarasov, V., and Zadok, E. (2017). To fuse or not to fuse: performance of user-space file systems. In *Proceedings of the 15th Usenix Conference on File and Storage Technologies*, FAST'17, page 59–72, USA. USENIX Association.

Vietinghoff, D., Kunkel, J., Kluge, M., Rahn, M., Krzikalla, O., Höhn, P., Falk, P., Zimmermann, P., Machado, R., and Oeste, S. (2024). Memory-Centric Storage for Exascale (MCSE).

Vietinghoff, D., Kunkel, J., Kluge, M., Rahn, M., Krzikalla, O., Höhn, P., Falk, P., Zimmermann, P., Machado, R., and Oeste, S. (2025). Memory-Centric Storage for Exascale (MCSE).

Vietinghoff, D., Kunkel, J., Kluge, M., Rahn, M., Krzikalla, O., Höhn, P., Falk, P., Zimmermann, P., Machado, R., and Oeste, S. (2026). Memory-Centric Storage for Exascale (MCSE).

3. Auflistung der erfolgten oder geplanten Veröffentlichungen des Ergebnisses.

Kunkel, J. (2024). Vi4io/io500-ior-mockup: An io500 mockup to showcase potential of new interfaces latest.

Oeste, S., Höhn, P., Kluge, M., and Kunkel, J. (2025). An analysis of the i/o semantic gaps of hpc storage stacks. *Frontiers in High Performance Computing*, Volume 3 - 2025.