Das parallele Dateisystem Lustre

Benedikt Johannes Riehm

Steinbuch Centre for Computing (SCC)
Karlsruher Institut für Technologie (KIT)
Prof. Dr. Achim Streit
Dr. Gevorg Poghosyan
Betreuer: Roland Laifer

Inhaltsverzeichnis

1 Einleitung

Eine typische Aufgabe innerhalb von vielen Forschungsprojekten stellt heutzutage die Durchführung von wissenschaftlichen Experimenten und Messungen dar, die mit Computersimulationen auf Hochleistungsrechnern gekoppelt werden. Hierdurch können zum einen vorhandene Ergebnisse validiert werden und zum anderen mögliche Resultate vorhergesagt werden. Auch können durch Simulationen neue Forschungsstrategien und -wege erkundet werden.
Bei derartigen Simulationen fallen große Datenmengen an, die vom System gelesen oder geschrieben werden müssen. Somit werden besonders hohe I/O-Anforderungen an das System gestellt.
Die hierfür benötigte Computerleistung wurde bis vor einigen Jahren maßgeblich durch Verbesserungen in der Mikroprozessortechnologie vorangetrieben. Die Speicherleistung hat sich demgegenüber nur langsam weiterentwickelt. Die so entstandene Lücke zwischen Rechen- und Speicherleistung wird immer größer, sodass nun die I/O-Performance den Flaschenhals ausmacht. Für das nun aufkommende Exascale-Computing müssen daher neue Speichertechnologien mit einer höheren Dichte und Bandbreite entwickelt werden.
Die meisten Programme sind I/O-blockierend. Somit ist die prozentuale Auslastung des Computerclusters stark an den I/O gebunden, da in der Wartezeit keine anderen Berechnungen ausgeführt werden können. Zusätzlich müssen Simulationen oft in einem absehbaren Zeitrahmen ausgeführt werden, damit die Ergebnisse sinnvoll verwendet werden können. Beispielsweise muss besonders bei Wettervorhersagen die Simulationszeit deutlich geringer als die Vorhersagezeit ausfallen.
Durch diese Anforderungen ist eine Parallelisierung des I/O's zusätzlich zu einer effizienten Parallelisierung des Programms unumgänglich geworden. Performante parallele Dateisysteme wie Lustre stellen somit eine Schlüsselkomponente in der Bewältigung dieses Rechenbedarfs dar.
[1] [2] [3]

2 Das parallele Datesystem Lustre

2.1 Historie, Status Quo

Die grundlegende Architektur des Dateisystems entstand aus einem Forschungsprojekt an der Carnegie Mellon University (CMU) unter der Leitung von Peter Braam im Jahre 1999. Dieser gründete hieraus zwei Jahre später das Unternehmen „Cluster File Systems, Inc“. 2003 wurde Lustre 1.0 unter einer Open-Source Lizenz erstmals veröffentlicht. Die Entwicklung von Lustre wurde gefördert durch eine Initiative des Department of Energy (DoE), bei der auch Hewlett-Packard und Intel dran beteiligt waren.
In den darauf folgenden Jahren ergaben sich weitreichende organisationale Veränderungen: Sun kaufte 2007 die Firma Clustre File Systems. 2010 wurde Sun selbst von Oracle übernommen. Kurz nach der Übernahme entschloss sich Oracle dazu, Lustre nicht weiter zu entwickeln. Dies gab den Anlass neue Organisationen zu gründen, die sich mit dem Support und der Weiterentwicklung von Lustre in einem „Open Community Development Model“ befassten. Hierzu zählen die Open Scalable File Systems, Inc. (OpenSFS), EUROPEAN Open File Systems (EOFS) und Whamcloud, die von OpenSFS zur Weiterentwicklung von Lustre beauftragt wurde. Nachdem Whamcloud den DoE-Auftrag zur Erweiterung Lustre's im Hinblick auf Exascale Computing gewonnen hatte, wurde es 2012 von Intel aufgekauft. Die High Performance Data Division (HPDD) innerhalb von Intel stellt seither den Hauptentwickler von Lustre dar. Neben Intel gibt es viele weitere Unternehmen (Cray, EMC, Suse) und National Labs (LLNL, ORNL, NASA), die an der Entwicklung von Lustre entweder durch neuen Quellcode oder durch Einreichung von Bugreports beteiligt sind.
Lustre ist ein inzwischen sehr stabiles, hoch verfügbares und hoch skalierbares Dateisystem. Es wird daher heutzutage in vielen Rechenzentren und Clustersystemen im Produktivbetrieb eingesetzt. Besonders hierbei im Bereich der Höchstleistungsrechner, wo es von über 70% der 100 schnellsten HPCs weltweit[1] genutzt wird. Die derzeit größten Lustre-Installationen umfassen mehrere 10.000 Clients mit einem Durchsatz von mehreren GB/s pro Client, sowie mehreren PBs an Gesamtstorage und einigen TB/s parallelem Gesamt-I/O. Getrieben durch die weite Verbreitung des Dateisystems haben sich viele Hard- und Softwareunternehmen auf Lustre spezialisiert und bieten umfangreiche Lösungen und Produkte hierfür an.
[3] [4]

[1] Liste der 500 schnellsten HPCs: `http://www.top500.org`

2.2 Architektur

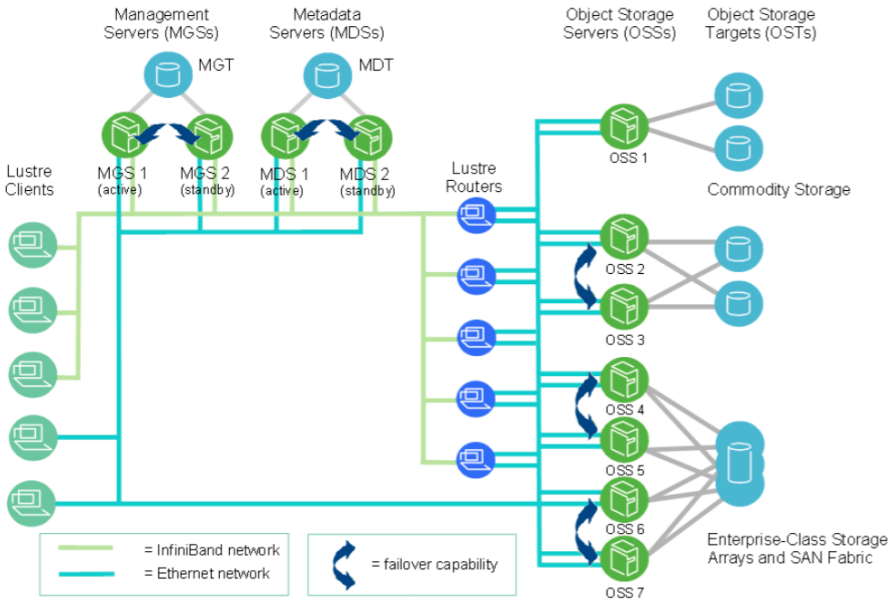

Abbildung 1. Architektur eines Lustre Clusters, Quelle: [5]

Eine Lustre-Installation besteht je nach Größe aus vielen verschiedenen Komponenten. In Abbildung 1 ist beispielhaft eine hochskalierte Cluster-Installation dargestellt, die den schematischen Grundaufbau visualisiert.
Der Managementserver (MGS) speichert allgemeine Konfigurationsinformationen über alle Dateisysteme in einem Cluster mit mehreren Lustre Dateisystemen. Jeder Lustre Dateiserver kontaktiert den MGS um Informationen über sich kundzugeben. Somit können angeschlossene Lustre Clients den MGS kontakieren, um etwa den Aufbau und Ort der Speichersysteme zu erfahren. Vorzugsweise sollte der MGS als eigenständiges System mit eigenem Speicher installiert werden um ihn unabhängig von anderen Komponenten managen zu können. Jedoch ist es auch möglich den MGS mit dem Metadatenserver (MDS) in einer Co-Location zu betreiben.
Der Metadatenserver (MDS) ist für die Metadaten im Dateisystem zuständig und stellt diese den Lustre Clients zur Verfügung. Somit ist der MDS unter anderem für die Verwaltung der Dateinamen und Ordnerstrukturen verantwortlich. Er regelt die Zuweisung der Speicherorte zu den entsprechenden Dateien. Auch findet in diesem Server ein Teil der POSIX-Konsistenzverwaltung sowie die Zugriffskontrolle der Daten statt.

Die eigentlichen Metadaten sind in dem Metadatentarget (MDT) abgespeichert. Der MDT ist Plattenspeicher („Block Device“), der an den Metadatenserver angeschlossen ist. Üblicherweise werden hierbei RAID10-Systeme installiert, um eine maximale Redundanz und Metadatenleistung zu erhalten. In ihm sind Dateinamen, Ordner, Berechtigungen und insbesondere das Dateilayout (Objekt IDs und OST Nummern, vgl. Kapitel 2.3) abgespeichert. Jedes Lustre Dateisystem hat einen MDT, auf dem die Metadaten liegen (seit Version 2.4 *mindestens* einen MDT).
Der Object Storage Server (OSS) ist für eine kleine Anzahl an Object Storage Targets (OST) verantwortlich und behandelt Netzwerkanfragen sowie den I/O-Zugriff der Lustre Clients zu den Nutzdaten auf diesen OSTs. Zwecks Caching speichert der OSS einige der Daten zwischen, für die er verantwortlich ist.
Das Object Storage Target (OST) ist analog zum MDT blockbasierter Plattenspeicher, der die eigentlichen Nutzdaten der User speichert. Ein OST ist dabei normalerweise über mehrere physikalische Platten in einem RAID-System mit Level 5 oder 6 realisiert, um eine hohe Geschwindigkeit und hohe Ausfallsicherheit zu gewährleisten. Die Dateien werden im Gegensatz zu anderen Dateisystemen (vgl. Kapitel 2.5) objektbasiert gespeichert: Eine Datei wird somit in ein oder mehrere Objekte aufgeteilt (vgl. Kapitel 2.4), die dann auf separate OSTs gespeichert werden.
Der Lustre Client ist üblicherweise ein Rechenknoten, der das Lustre Dateisystem benutzen und Dateien lesen beziehungsweise schreiben möchte. Auf ihm ist die Lustre Client-Software installiert, womit dieser mit dem Lustre Dateisystem kommunizieren kann. Die Client-Software ermöglicht das Mounten des Dateisystems und stellt ein Interface zwischen dem Lustre Virtuellen Dateisystem und den Lustre Servern bereit. Neben dem Metadatenclient (MDC) und Managementclient (MGS) gibt es für jeden OSS einen Objekt Storage Client (OSC). Die OSCs werden zu einem Logical Object Volume (LOV) aggregiert, um einen transparenten Zugriff auf die OSTs sicherzustellen. Der Lustre Client sieht hierdurch einen einheitlichen, kohärenten und synchronisierten Namensraum.
Die letzte wichtige Komponente in einer Lustre Installation stellt das Lustre Networking (LNET) dar. Hier ist die Netzwerkkommunikation zwischen den Lustre Clients und Servern implementiert. Das LNET unterstützt viele verschiedene Netzwerktypen: InfiniBand, TCP (GigE/10GigE/IPoIB), Cray, Myrinet, RapidArray, Quadrics. Auch stellt LNET das Routing zwischen verschiedenen Netzwerktypen bereit, die gleichzeitig in einem Lustre Cluster verwendet werden können. Falls der eingesetzte Netzwerktyp Remote Direct Memory Access unterstützt, so stellt LNET dies den Clients und Servern zur Verfügung.
In Lustre gibt es je nach Version zwei Ausbaustufen für Server-Failover: Aktiv-Passiv-Failover (A/P-Failover) typischerweise für den MDT sowie einen Aktiv-Aktiv-Failover (A/A-Failover) typischerweise für die OSTs. Beim Aktiv-Aktiv-Failover sind beide Server aktiv und stellen Teilmengen der Ressourcen zur Verfügung. Falls einer der beiden Server ausfällt, übernimmt der andere Server seine Ressourcen und stellt sie dem Lustre-System bereit. Der Aktiv-Passiv-Failover ist im Prinzip gleich aufgebaut, alleine ist hierbei die Ressourcenteilmenge leer.

Somit gibt es einen Server, der die aktive Rolle der Dienstbereitstellung mitsamt allen Ressourcen übernimmt. Der zweite Server fungiert lediglich als passiver Stand-by-Server und übernimmt die Dienstbereitstellung im Falle eines Ausfalls des aktiven Servers.

Lustre unterstützt bis Version 2.4 für den MDS nur ein Aktiv-Passiv-Failover. Oft übernimmt der Stand-By-Server jedoch die Rolle des MGS oder eines aktiven MDS von einem anderen Lustre Dateisystem, um die Hardwareressourcen optimal ausnutzen zu können. Die OSSs können als Aktiv-Aktiv-Failover realisiert werden um Redundanz ohne größeren Overhead zu erhalten. Ab Version 2.4 wurde mit dem Distributed Namespace (DNE) eine Funktionalität zu Lustre hinzugefügt, die es erlaubt MDTs für individuelle Unterordner eines Lustre Dateisystems zu realisieren. Dadurch sind auch Aktiv-Aktiv-Failoverinstallationen für MDSs verfügbar.
Im Szenario eines Server-Failovers (MGS, MDS oder OSS) versucht der Client eine I/O-Anfrage zu dem ausgefallenen Server zu senden und probiert dies so lange bis ein Failover-Server die Ressourcen des defekten Servers übernommen hat und dem Client eine Antwort zurücksendet. Die Failover-Behandlung passiert dabei allein auf Ebene des Lustre-Kernelmoduls, sodass ein Failover nach einem Serverausfall bis auf längere Wartezeiten für Benutzeranwendungen vollkommen transparent erscheinen.
In Tabelle 1 sind zum einen die theoretischen Begrenzungen des Lustre Dateisystems aufgelistet. Zum anderen wird das zum Stand 2014 größte installierte Lustre-System mit dessen Begrenzungen präsentiert. Hierbei zeigt sich, dass mit Lustre zumindest aus theoretischer Sichtweise noch deutlich größere und schnellere Installationen möglich wären. Aus praktischer Sicherweise sind jedoch oft Kosten, Kühlsysteme, Systemkomplexität oder Benutzerverhalten begrenzende Faktoren, die es zu lösen gilt.
[5] [6] [7]

Tabelle 1. Theoretische Begrenzungen & Produktivsystem, Quelle: [8]

Lustre Features	**Theoretische Begrenzung**	**Größtes Produktivsystem** [2]
Dateisystemgröße	512 PB	55 PB
Anzahl Dateien	4 mrd. pro MDT	≈ 2 mrd.
Größe einer Datei	2.5 PB	100 TB
Aggr. Performance	7 TB/s	1.1 TB/s
Anzahl Clients	>100.000	≈ 50.000

2.3 Datenstruktur und Datei-I/O

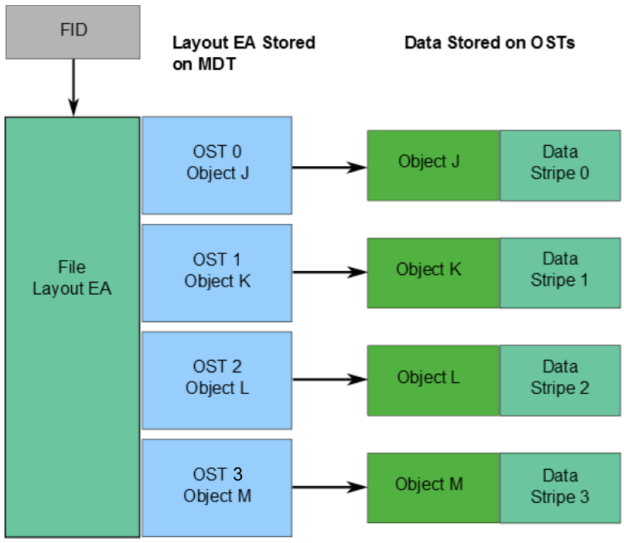

Abbildung 2. Aufbau der Inodes in einem Lustre-System, Quelle: [5] (editiert)

Der Speicherort einer jeden Datei ist in Lustre als ein Extended Attribute (EA) auf dem Metadatenserver abgespeichert und wird über den zugehörigen Lustre File Identifier (FID) referenziert (vgl. Abbildung 2). Die FIDs ersetzen dabei die sonst verwendeten UNIX Inode Numbers. Der File Layout EA beinhaltet Zeiger auf 1-bis-N Objekte und die zugehörigen OSTs, in denen die Dateiobjekte abgespeichert sind. Hiermit kann nachvollzogen werden, auf welchem OST welches Objekt einer Datei liegt.
In der Abbildung 2 ist eine Datei in vier Objekte unterteilt und auf vier verschiedenen OSTs gespeichert worden. OST 0 hält das erste Objekt (Objekt J), OST 1 das zweite Objekt (Objekt K), OST 3 das dritte und OST 4 das letzte Objekt der Datei. Jedes dieser Objekte beinhaltet dabei mindestens ein File-Stripe (vgl. Kapitel 2.4) der gesamten Datei.

In Abbildung 3 ist ein Schreibvorgang einer Datei exemplarisch abgebildet. Der Lustre Client möchte hierbei eine Datei mit einem vorher festgelegten Stripe Count über mehrere OSTs parallel schreiben. Hierfür muss der Client zunächst eine Dateiöffnungsanfrage als RPC-Request an den Metadatenserver schicken, um insbesondere das Dateilayout zu erhalten. Der Metadatenserver antwortet dem Client mit dem Lustre File Identifier als RPC-Response, in der das Objekt-OST-Mapping beinhaltet ist. Falls die Datei noch nicht im Dateisystem existiert, weist der Metadatenserver der neuen Datei respektive seiner Objekte eine oder mehrere OSTs zu, welche per default round-robin ausgewählt werden. Nach Erhalt

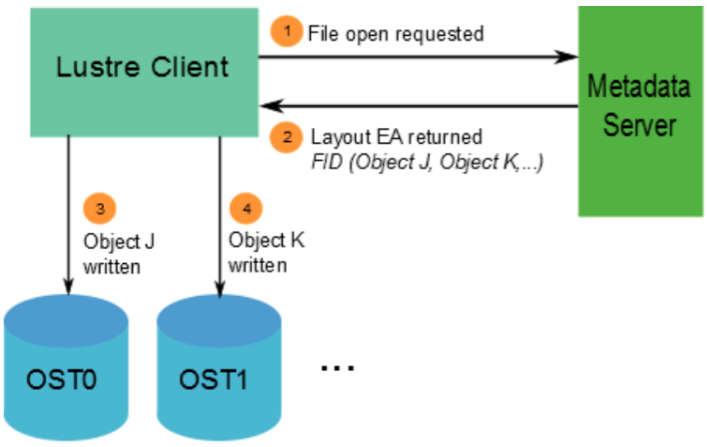

Abbildung 3. Datei-I/O in einem Lustre-System, Quelle: [5]

des Dateilayouts führt der Client nun die einzelnen Schreibvorgänge nur noch mit den jeweiligen OSSs durch, die für die OSTs verantwortlich sind. Hierbei findet die eigentliche I/O-Kommunikation wiederum über RPC-Aufrufe statt. Die OSSs regeln dabei über das Lustre Locking-Management den kohärenten Dateizugriff.
[5]

2.4 File Striping

Das File Striping ist einer der Hauptfaktoren für die hohe Performance und stellt somit ein integrales Konzept in der Lustre Architektur dar. Bei dem File Striping werden die Dateien in kleine Segmente, sogenannte Stripes, aufgeteilt und auf ein oder mehrere OSTs standardmäßig nach dem Round-Robin-Verfahren verteilt. Wesentliche Parameter des File Stripings sind zum einen die Stripe Size, welche die Größe der einzelnen Segmente angibt, in die die Datei unterteilt werden soll. Zum anderen definiert der Stripe Count die Anzahl an Objekten und damit die Anzahl an OSTs, auf die die Datei aufgeteilt werden soll. Jedes Objekt beinhaltet dabei ein oder mehrere Stripes derselben Datei. Um einen kohärenten Dateizugriff gemäß POSIX-Standard zu ermöglichen, müssen die Clients vor dem Datei-I/O ein Object-Lock vom dazugehörigen OSS für jedes benötigte Objekt anfordern. Andere Clients, die auf die gleichen Objekte zugreifen wollen, werden vom OSS blockiert, bis der aktive Client fertig gelesen beziehungsweise geschrieben hat. Für ungeschriebene Daten wird kein Speicher auf den OSTs reserviert. Ein weiterer Parameter ist der Stripe Offset. Dieser gibt an, bei welchem OST mit dem Striping der Datei begonnen werden soll. Der Stripe Offset sollte üblicherweise nie vorgegeben sondern dem System überlassen werden, da es sonst zu einer ungleichmäßigen Ausnutzung von einigen OSTs kommen kann.

Abbildung 4. Datei-Striping auf drei OSTs, Quelle: [9]

Abbildung 4 verdeutlicht hierbei das Konzept exemplarisch anhand von drei Dateien, die mit unterschiedlichen Lustre Striping Parametern auf drei verschiedene OSTs gestriped wurden. Datei A ist 9 Megabyte groß und wurde mit der default Stripe Size von 1 Megabyte und einem Stripe Count von 3 im Lustre Dateisystem vorkonfiguriert. Durch die Stripe Size wird die Datei in 9 gleichgroße Stripes zu je 1 Megabyte unterteilt. Der Stripe Count bewirkt, dass die 9 Stripes auf drei OSTs verteilt werden. Der Offset ist hierbei auf Null gesetzt, sodass bei dem ersten OST angefangen wurde zu stripen. Die Datei B ist 1 Megabyte groß und wurde analog wie Datei A mit der default Stripe Size von 1 Megabyte konfiguriert. Der Stripe Count ist auf 1 festgelegt worden, da die Datei in ein Stripe passt. Der Offset wurde auf den zweiten OST gesetzt. Somit wurde die Datei in ein Stripe unterteilt und in ein Objekt auf OST_1 geschrieben. Datei C ist 2 Megabyte groß, wurde jedoch mit einer Stripe Size von 2 Megabyte und folglich einem Stripe Count von 1 konfiguriert. Der Stripe Offset wurde auf den dritten OST gesetzt. Sie wird daher als ein Stripe in einem Objekt auf OST_2 geschrieben.

Das File Striping bringt einige Vorteile, aber auch Nachteile mit sich. Durch die Aufteilung der Daten in kleinere Segmente und das anschließende Verteilen der Segmente auf mehrere OSTs entsteht paralleler Datei-I/O, der einen höheren Datendurchsatz gegenüber dem I/O mit nur einem Storage-Target zur Folge hat. Ein weiterer Vorteil ist die Möglichkeit, eine Datei zu speichern, die die Kapazität eines einzelnen OST übersteigt, da der Speicherplatz nun der Summe aller OSTs entspricht.
Das Striping hat jedoch den Nachteil, dass es mehr Overhead verursacht. Dieser

Overhead entsteht zum einen durch die zusätzlichen Locks, die für die Objekte nötig sind. Bei falschen Zugriffsmustern („Nicht-ausgerichtetes Striping“, vgl. Kapitel 3.2) haben die resultierenden Lockingkonflikte einen deutlichen Performanceverlust zur Folge. Zum anderen entsteht der Overhead durch mehr RPCs, die das Netzwerk höher belasten, und durch eine höhere Belastung der OSSs selbst, da die Clients viele Objekt-I/O-Anfragen an sie schicken und gegenseitig um die Ressourcen konkurrieren. Auch ergibt sich durch das Striping ein höheres Dateiverlustrisiko, was mit dem Stripe Count der Dateien ansteigt.
[5]

2.5 Vergleich zu anderen parallelen Dateisystemen

Neben Lustre gibt es noch einige andere parallele Dateisysteme. Die wichtigsten Vertreter, die auch in der Industrie und Wissenschaft eingesetzt werden, sind das General Parallel File System (GPFS) von IBM und das pNFS. Alle drei Dateisysteme sind dadurch gekennzeichnet, dass sie die Metadaten von den Nutzdaten trennen und durch mehrere Dateiserver parallelen I/O auf den Nutzdaten ermöglichen.
Das Dateisystem GPFS ist gekennzeichnet durch seinen größeren Funktionsumfang gegenüber Lustre, der im Dateisystem integriert ist. So unterstützt es beispielsweise Snapshots, Metadaten- sowie Nutzdatenreplikation und besitzt ein integriertes Lifecycle-Management. Es läuft ähnlich wie aktuelle Lustre-Versionen sehr stabil, skaliert jedoch nur bis zu einigen Tausend Clients und damit eine Größenordnung geringer als Lustre. Es bietet nativen Support auf vielen verschiedenen Betriebssystemen, unter anderem hierbei AIX, Linux, Windows Server sowie das IBM-eigene zOS der zSeries Plattform.
Während Lustre eine Open-Source Softwarelösung ist und viele Hardwarehersteller hierzu Hard- und Software-Produkte anbieten, ist GPFS ein rein proprietäres Dateisystem eines einzigen Herstellers. Dabei ist insbesondere die beschränkte Hardware-Auswahl problematisch, die man als Nutzer zur Verfügung gestellt bekommt. GPFS unterstützt beispielsweise als Netzwerkprotokolle nur IP, Infiniband und Federation. Auch hat man als Kunde bei dieser Lösung einen Vendor-Lock-in, der sich durch die häufigen Lizenzänderungen von IBM weiter verstärkt. Ferner ist das GPFS nicht im Linux-Kernel enthalten, sodass bei jeder neuen Kernelversion Anpassungen der GPFS-Module getätigt werden müssen.
Das Dateisystem Parallel NFS (pNFS) ist als Version 4.1 des NFS Standards definiert. Hierbei existieren mit Files, Blocks und Objects drei verschiedene Implementierungen, die eingesetzt werden können. Durch die Standardisierung und den offenen Quellcode gibt es, ähnlich zum Open-Source-Ansatz von Lustre, viele Hardwarehersteller wie NetApp, EMC oder Panasas, die Server für pNFS anbieten. Clientseitig ist pNFS vollständig im Linux-Kernel ab Version 3.1 und in Windows ab Version Vista und Server 2008 implementiert.
Die Hauptnachteile von pNFS sind zum einen die sehr begrenzte Skalierbarkeit von wenigen Hundert Clients, die den Einsatz in großen Installationen verhindert. Auch ist pNFS durch seine Close-to-Open-Semantiken nicht gut für HPC-Anwendungen mit parallelem IO geeignet. Zum anderen gibt es Probleme bei

der Stabilität des Dateisystems. Nicht zuletzt durch die drei unterschiedlichen Implementierungen entsteht ein sehr kompliziertes Produkt, was sich negativ auf die Stabilität des Systems auswirkt. Auch muss bei neuen Installationen noch mit vielen Bugs gerechnet werden.
[4] [2] [10]

3 Lustre im Praxiseinsatz

3.1 Aspekte auf Benutzerseite

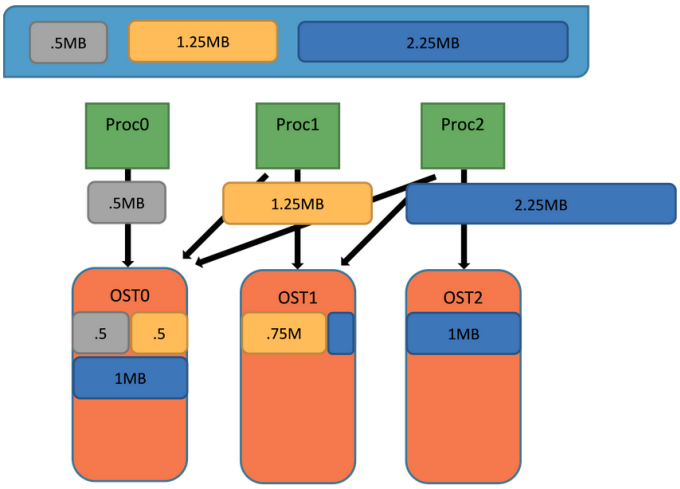

Abbildung 5. Beispiel von nicht-ausgerichtetem Striping, Quelle: [9]

Ausgerichteter I/O beim File-Striping (vgl. Kapitel 2.4) ist eine wichtige Voraussetzung für eine optimale Ausnutzung des Lustre Dateisystems. Dabei ist ein File-Striping „ausgerichtet“, falls man auf dies an Offsets zugreifen kann, die mit den Stripe-Grenzen übereinstimmen.

In Abbildung 5 wird die Problematik des Nichtausrichtens der File-Stripes beispielhaft verdeutlicht. Eine 4 Megabyte große Datei soll von 3 Prozessen auf 3 unterschiedliche OSTs gestriped werden. Die Stripe Size ist wie in Kapitel 2.4 auf 1 Megabyte festgelegt, der Stripe Count ist 3 (da auf 3 OSTs geschrieben werden soll). Die Prozesse teilen jedoch die Datei nicht auf Stripes auf, die an die Stripe-Grenzen ausgerichtet sind: Prozess 0 schreibt die ersten 0.5 Megabyte der Datei auf OST 0, Prozess 1 schreibt die nächsten 1.25 Megabyte, angefangen bei Offset 0.5 Megabyte. Dieser Datei-I/O muss von Lustre aufgrund der gewählten Stripe Size aufgesplittet werden und wird dann auf das angefangene erste Stripe

und ein weiteres Stripe auf dem zweiten OST geschrieben. Analog hierzu wird der 2.25 Megabyte große I/O von Prozess 2 auf 3 verschiedene Teilchuncks aufgesplittet und auf die OSTs verteilt.
Der gesamte Schreibvorgang verursacht 6 Schreib-Anfragen der Clients. Wäre die Datei auf 1 Megabyte große Stripes aufgeteilt, würde der Schreibvorgang lediglich 4 IO-Anfragen verursachen. Zusätzlich müssen die Clients ein Lock für die jeweiligen Objekte akquirieren, um auf die OSTs zu schreiben. Bei dem ersten Stripe (Prozess 0 und Prozess 1) und bei dem zweiten Stripe (Prozess 1 und Prozess 2) gibt es daher einen Konflikt um die jeweiligen Objekt-Locks, was durch die Serialisierung zu weiteren Verzögerungen und Ressourcenstau auf den Servern führt. Diese Besonderheit bei parallelem Datei-I/O in Lustre muss auf Benutzerseite berücksichtigt werden, da sonst sowohl die Benutzeranwendung als auch das Gesamtsystem (OSSs/OSTs) Performanceverluste erleidet.

Neben dem Ausrichten der Stripes sollte der Nutzer in einem Lustre-System noch einige weitere Dinge beachten, um das Dateisystem optimal zu nutzen und keine unnötigen IO-Anfragen zu stellen, die zu Verschlechterungen der Performance führen: Zum einen sollten Nutzer ihre Dateien nur im Lesemodus öffnen und vor dem Schreiben die Daten zwischenpuffern, um keine unnötigen Locks zu belegen. Bei MPI-parallelisierten Programmen sollte der I/O über (wenige) dedizierte I/O-Prozesse abgewickelt werden, die die Daten per MPI-Broadcast nach dem Lesen an die anderen Prozesse verteilen und die zu schreibenden Daten mit MPI-Collect von den anderen Prozessen einsammeln. Nutzer sollten des weiteren einen zur Dateigröße angemessenen Stripe Count wählen. Kleine Dateien sollten dabei nur auf ein OST geschrieben werden, größere Dateien sollten auf mehrere OSTs gestriped werden. Allgemein sollte jedoch das Lesen, Schreiben und Löschen von vielen kleinen Dateien vermieden werden, um den MDS, OSS und das Lustre Netzwerk durch den kleinen Payload in den RPCs und die notwendigen Locks nicht unnötig zu belasten. Insbesondere sollten keine großen Softwarepakete kompiliert werden, die viele kleine Objektartefakte während des Kompiliervorgangs erzeugen. Auch sollten keine Bibliotheken oder Frameworks im Lustre Dateisystem installiert werden, die viele kleine Dateien beinhalten und unter Umständen von vielen Clientanwendungen im Gesamtsystem als Abhängigkeit angefordert werden.
Um den zentralen Metadatenserver nicht unnötig mit Metadatenanfragen zu fluten, sollte das Ausführen von Befehlen unterbunden werden, die eine große Zahl an Metadaten anfordern. Hierzu zählen beispielsweise rekursives GNU `find` und `ls`, aber auch Befehle wie `rm tar` mit dem Einsatz von Wildcards jeweils auf großen Ordnerstrukturen. Bei GNU `ls` werden dabei Metadaten (Besitzer, Rechte, Erstellungszeiten) vom MDS und die Dateigröße von den zugehörigen OSTs angefordert. In Befehlen mit Wildcards bei großen Ordnerstrukturen benötigt die Evaluierung dieser Wildcard ebenfalls wertvolle Rechenzeit und sollte daher wenn möglich nicht benutzt werden.

3.2 Aspekte auf Betreiberseite

Insbesondere bei dem Betrieb von großen Clustersystemen haben sich in den letzten Jahren zwei Aspekte entwickelt, die heutzutage aus Sicht der Systembetreiber ernstzunehmende Probleme darstellen:
Die „Silent Data Corruption“ beschreibt den Umstand, dass Daten im Dateisystem verändert werden, ohne das Betreiber respektive Benutzer hierüber vom Betriebssystem durch eine Fehlermeldung in Kenntnis gesetzt werden. Die Ursachen für ein derartiges Fehlverhalten des Systems können äußerst vielfältig sein. Zum einen kann die darunterliegende Hardware defekt sein. Dabei müssen alle in der Verarbeitungskette beteiligten Komponenten mitberücksichtigt werden, angefangen bei der CPU, dem Hauptspeicher, den Festplatten und eventuellen RAID-Controllern bis hin zu den Interfaces, Routern und Switches des Clusternetzwerks. Auch die Firmware auf den RAID-Controllern, Festplatten oder Netzwerkkomponenten kann aufgrund der immer höheren Komplexität fehlerhaft sein. Nicht zuletzt kann das Betriebssystem mit seinen Subkomponenten Virtuelle Speicherverwaltung, Dateisystem, Systembibliothek oder Festplattentreiber Bugs beinhalten.
Das Problem der Silent Data Corruption wurde bis vor einiger Zeit nur äußerst selten im Betrieb von großen Installationen beobachtet. Dies hat sich mit der immer stärker ansteigenden Menge an Daten, die verarbeitet und gespeichert werden, in den letzten Jahren geändert und stellt heutzutage ein reales Problem dar. So zeigte eine Studie des CERN [11], dass jede 1500. Datei in deren Dateisystem korrupt war. Dies entspricht einer Bytefehlerrate des Gesamtsystems von $1 \cdot 10^{-7}$ und weicht damit signifikant von üblichen Bytefehlerraten ($1 \cdot 10^{-14}$) ab, die Hersteller für Storagesubsysteme als Angabe treffen.
Um der Silent Data Corruption entgegenzuwirken, können insbesondere zwei Maßnahmen getroffen werden. Zum einen können Ende-zu-Ende-Checksummen und Korrekturfunktionen in das Gesamtsystem implementiert werden, die Änderungen an den Daten zwischen Quelle und Ziel der Übertragungskette registrieren und korrigieren. Zum anderen können mehrere Kopien derselben Datei im Dateisystem angelegt werden um eine niedrigere Fehlerrate der Dateien zu erhalten. Beide Maßnahmen haben jedoch Komplexität, Performanceverlust und Speicherverbrauch zur Folge. Ihr Einsatz ist jedoch im Hinblick auf die möglichen Verluste an Nutzdaten empfehlenswert.
Der „Triple Disk Failiure“ stellt ein zweites wichtiges Problem auf Betreiberseite dar. In einem Lustre-System sind die OSTs üblicherweise als RAID-Array mit RAID-Level 5 oder 6 realisiert (vgl. Kapitel 2.2). Bei RAID-Level 6 hat der Ausfall von 3 Festplatten einen Datenverlust zur Folge. Im Falle eines Festplattenausfalls muss sich das RAID-System nach dem Plattentausch durch den Vorgang des RAID-Rebuildings neu aufbauen. Dies verursacht einen höheren Workload des RAID-Systems und erhöht automatisch die Ausfallwahrscheinlichkeit der anderen Festplatten im RAID-Verbund. Im schlimmsten Fall kann dies zum sukzessiven Ausfall aller drei Festplatten führen, noch bevor das RAID-Rebuilding abgeschlossen ist. Dieses Ereignis wird aufgrund seines relativ häufigen Auftretens in großen Clustersystemen und den schwerwiegenden Folgen als „Triple Disk

Failiure“ bezeichnet.
Das Problem des Triple Disk Failiure kann auf zwei Weisen gelöst werden. Zum einen können die OSTs gespiegelt in einem Failover-Betrieb arbeiten. Dies wird in der aktuellen Lustre-Version jedoch noch nicht automatisch unterstützt, sodass die Systembetreiber diese Spiegelung manuell durchführen müssten. Zum anderen kann ein Software-RAID über die RAID-Systeme der OSS gebaut werden, um eine höhere Ausfallsicherheit zu erhalten. Dies ist jedoch unweigerlich mit Performanceverlusten und einer höherer Systemkomplexität verbunden.

4 Performance-Analyse

Lustre eignet sich durch seine Architektur für verschiedene IO-Aufgaben unterschiedlich gut. In Kapitel 3.1 wurden bereits einige Best Practices im Umgang mit dem Dateisystem dargelegt und insbesondere Fälle beschrieben, die für Lustre ungeeignet sind. Ein Benutzer sollte daher immer den Einsatz von Lustre statt lokalen Plattenspeichern bei seinen Anwendungen abwägen. Um den Unterschied von Lustre und lokalem Speicher in Hinblick auf IO-Performance aufzuzeigen, wurde eine Analyse auf dem bwUniCluster HPC-System des Steinbuch Centre for Computing (SCC) des Karlsruher Instituts für Technologie (KIT) durchgeführt.

4.1 Setup

Die Performance-Analyse wurde auf zwei Login-Knoten des HPCs im laufenden Betrieb durchgeführt. Die relevanten Systemeigenschaften können der Tabelle 2 entnommen werden. Die Knoten haben dabei lokale Festplatten in Form eines RAID-5-Systems und sind an das globale SCC-Lustre mit Infiniband angebunden. Die zwei genutzten OSTs bestehen jeweils aus RAID-6-Systemen.
Die Analyse wurde in drei Kategorien unterteilt, die verschiedene Aspekte der IO-Performance bestimmen: Datendurchsatz, I/O-Operationen pro Sekunde (IOPS) sowie Metadatenleistung. Die ersten beiden Kategorien wurden mithilfe der Tools `GNU dd` und `fio` gemessen, die auf eigens erzeugten Testdaten ausgeführt wurden. Die Metadatenleistung wurde mit dem Tool `GNU ls` ermittelt, welches auf dem derzeit aktuellsten Linux-Kernel-Quellcode-Ordner rekursiv dessen Metadaten aufgelistet hat.

Tabelle 2. Performance-Setup

Eigenschaft	**Login-Knoten**
Prozessor	Intel Xeon E5-2670, 16 Kerne @ 2.6GHz
Leistung	332,8 GFLOPS
Hauptspeicher	64 GB
Netzwerk	Infiniband 4X FDR
Festplatten	RAID-5 (5 x 1TB)
OSTs	DDN SFA12K RAID-6
Betriebssystem	Red Hat Enterprise Linux 6.4
Lustre	Lustre 2.5
Durchsatz	Block Size: 1M, Stripe Count: 2, 100GB
IOPS	Block Size: 4K, Stripe Count: 2, 1GB
Metdaten	`ls -lR /linux-kernel-src`, Stripe Count: 2

4.2 Ergebnisse

Tabelle 3. Ergebnisse: Lokale HDDs vs. Lustre

System	Durchsatz	IOPS	Metadaten
Lokal	430 MB/s	900 r | 320 w	0.7 s
Lustre	1.7 GB/s	165 r | 55 w	6.2 s

Die gewonnenen Messresultate können der Tabelle 3 entnommen werden. Aufgrund der Messung im laufenden Betrieb dürfen diese Ergebnisse lediglich als ungefähre Richtwerte aufgefasst werden. Es lässt sich jedoch dabei erkennen, dass ein paralleles Dateisystem wie Lustre für parallelen IO mit größeren Blockgrößen (1 Megabyte) besser geeignet ist als lokale Platten. Lustre war so beim gemessenen Datendurchsatz etwa 4 Mal schneller.
Bei der IOPS- und Metadatenmessung hat sich jedoch auch gezeigt, wofür Lustre weniger gut geeignet ist. Wie zuvor bereits erwähnt (vgl. Kapitel 3.1), sollten Nutzer es vermeiden, viele kleine Dateien zu lesen oder zu schreiben sowie viele Metadatenanfragen an den MDS und die OSSs zu senden. Da die meisten Lustre-Operationen systemseitig auf 1MB-großen Blöcken arbeiten, entsteht bei Nutzdaten mit 4K-Blockgrößen viel Overhead. Auch kommt es zu vielen Lock-Konflikten, da viele 4K-Blöcke auf das gleiche Objekt geschrieben werden. Bei

der Metadatenleistung ist der zentrale Metadatenserver die begrenzende Komponente. Die Messungen bestätigen hierbei diese Theorie. So waren die lokalen Festplatten bei der IOPS-Messung cirka 5 Mal schneller und bei der Messung der Metadatenleistung sogar fast 10 Mal schneller als das Lustre-System.

5 Fazit

Durch die immer schneller ansteigende Menge an Daten stellen performante parallele Dateisysteme eine Schlüsselkomponente dar. Sie sind eine wichtige Voraussetzung zur Erreichung der Exascale-Ära in den kommenden Jahren. Ein solches Dateisystem stellt Lustre dar, welches durch hohe Marktanteile von über 70% einen Quasi-Standard im HPC-Bereich definiert. Durch die komplexe Architektur müssen Benutzer des Dateisystems grundlegendes Wissen über Lustre haben, um ihre Anwendung performant ausführen zu lassen und die Systembelastung gering zu halten. Die anfallenden Datenmengen und Systemgrößen haben dabei in den letzten Jahren zu neuen Problemen wie der Silent Data Corruption oder dem Triple Disk Failiure geführt und müssen mit entsprechenden Gegenmaßnahmen und zusätzlichem Aufwand gelöst werden. Die im Laufe der Arbeit getätigte Performance-Analyse hat die zuvor beschriebenen Best Practices bestätigt. Hierbei zeigten sich Stärken (Durchsatz großer Dateien) sowie Schwächen (viele kleine Dateien, viele Metadatenoperationen) des Dateisystems. Somit muss der Benutzer durch sein Wissen selbstständig entscheiden, wo seine Daten und IO-Anfragen optimal im System verarbeitet werden können (lokal oder OSTs) und welche Einstellungen (Stripe Count, Stripe Size, uvm.) seinen Anforderungen am besten genügen.

Literatur

1. NERSC. (2012) Challenges in HPC. [Online]. Available: https://www.nersc.gov/assets/pubs_presos/ChallengesInHPC-June-2012.pdf

2. M. Kluge, "Comparison and end-to-end performance analysis of parallel file systems," Ph.D. dissertation, Technische Universität Dresden, 2011.

3. Wikipedia. (2015) Lustre (file system). [Online]. Available: https://en.wikipedia.org/wiki/Lustre_(file_system)

4. R. Laifer. (2011) Lessons learned from parallel file system operation. [Online]. Available: https://www.scc.kit.edu/scc/docs/Lustre/kit_elws2011_20110926.pdf

5. Intel. (2015) Lustre Software Release 2.x: Operational Manual. [Online]. Available: http://build.whamcloud.com/job/lustre-manual/lastSuccessfulBuild/artifact/lustre_manual.pdf

6. Sun microsystems. (2007) Lustre File System - High Performance Storage Architecture and Scalable Clustre File System (White Paper). [Online]. Available: http://www.csee.ogi.edu/~zak/cs506-pslc/lustrefilesystem.pdf

7. University of Tenessee. (2015) I/O and Lustre Usage. [Online]. Available: https://www.nics.tennessee.edu/computing-resources/file-systems/io-lustre-tips

8. T. K. P. (Seagate). (2015) Inside The Lustre File System (Technology Paper). [Online]. Available: http://www.seagate.com/files/www-content/solutions-content/cloud-systems-and-solutions/high-performance-computing/_shared/docs/clusterstor-inside-the-lustre-file-system-ti.pdf

9. University of Colorado Boulder. (2015) Parallel IO on Janus Lustre. [Online]. Available: https://rc.colorado.edu/support/examples-and-tutorials/parallel-io-on-janus-lustre.html

10. R. H. Dean Hildebrand, Arifa Nisar. (2009) pNFS, POSIX, and MPI-IO: A Tale of Three Semantics. [Online]. Available: http://www.pdsi-scidac.org/events/PDSW09/resources/pdsw09-final4.pdf

11. B. Panzer-Steindel. (2007) Data integrity. [Online]. Available: http://indico.cern.ch/event/13797/session/0/contribution/3/attachments/115080/163419/Data_integrity_v3.pdf

12. W. Schön. (2009) Lustre at GSI - Current Status and Outlook. [Online]. Available: http://www-alt.gsi.de/documents/DOC-2009-Sep-119-1.pdf

13. NASA. (2014) Lustre Best Practices. [Online]. Available: http://www.nas.nasa.gov/hecc/support/kb/Lustre-Best-Practices_226.html

14. NASA. (2015) Lustre Basics. [Online]. Available: http://www.nas.nasa.gov/hecc/support/kb/Lustre-Basics_224.html