Lustre

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

Vorgelegt von: Julius Plehn
E-Mail-Adresse: juplehn@me.com
Matrikelnummer: 6535163
Studiengang: Software System Entwicklung

Betreuer: Michael Kuhn

Hamburg, den 22.09.2016

Contents

1 Bedarf von Lustre

Lustre ist ein verteiltes Dateisystem. Dies spielt bei den Anwendungszwecken eine Rolle, in denen die Ressourcen eines einzelnen Servers nicht ausreichend sind. Besonders begrenzend ist die Geschwindigkeit der Festplatten und die Übertragungsgeschwindigkeit im Netzwerk. Eine HDD kann ca. 200 MB/s schreiben, während eine SSD ca. 500 MB/s erreicht. Ein RAID 0 System kann diese Geschwindigkeit etwa verdoppeln, indem Daten auf mehrere Festplatten aufgeteilt werden. Im Idealfall wird die Geschwindigkeit des Speicherverbundes nur durch die Schnittstelle limitiert. Allerdings stößt man auch hier schnell an nicht überwindbare Grenzen. Die SAS-Schnittstelle erreicht zum Beispiel maximal 22,5 GBit/s[SAS16]. Ähnlich limitierend ist auch die Netzwerkanbindung. Ethernet erreicht eine Geschwindigkeit von ca. 40 GBit/s, während Infiniband 100 GBit/s erreicht.

Diese Probleme werden von Lustre weitestgehend dadurch gelöst, dass mehrere Server mit den jeweiligen genannten Beschränkungen zu einem großen System verbunden werden. Dabei wird ein ähnliches Verfahren wie bei einem RAID 0 Festplattenverbund genutzt. Dateien werden durch sogenanntes Striping aufgeteilt und die Limitierungen eines einzelnen Systems entfallen.

Diese Verteilung ermöglicht aktuell reale Bandbreiten von ca. 2,5 TB/s (theoretisch 10 TB/s). Zudem werden aktuell schon über 55 Petabyte in einer einzigen Installation gespeichert (max. 512 PB). Der Datenzugriff erfolgt dabei durch das herkömmliche POSIX Interface. Des weitern ist Lustre durch eine Open Source Lizenz frei verfügbar. Das ist unteranderem der Grund, weshalb Lustre laut der TOP500 Liste von 60 der schnellsten 100 Hochleistungsrechnern verwendet wird.

Im Folgenden werde ich auf wichtige Konzepte und Vorteile von Lustre eingehen. [**?**]

2 Konzepte

2.1 Komponenten

Lustre unterscheidet im wesentlichen 4 verschiedenen Komponenten (s. Figure 2.1). Dabei wird besonders unter Meta Servern und Object Storage Servern unterschieden. Während die Meta Server die Dateien, Ordner und Rechte etc. verwalten, organisieren die anderen Server den eigentlichen Schreib- und Leseprozess.

Metadata Target (MDT):
Der MDT verwaltet die Metadaten. Dazu gehören neben den Namen auch Informationen zu den Stripes und Locks.
Metadata Server (MDS):
Der MDS stellt die von den MDT gespeicherten Metadaten den Clients zur Verfügung. Nach dem Abruf dieser Informationen ist der MDS nicht am eigentlichen IO beteiligt.
Object Storage Target (OST):
Die Daten werden in unterschiedlichen Objekten gespeichert. Diese werden auf verschiedenen OSTs gespeichert. Ein einzelner OST kann bis zu 128 TB speichern.
Object Storage Server (OSS):
Ein OSS verwaltet den Zugriff auf bis zu 32 OSTs.

2.2 Dateisysteme

Lustre ermöglicht den Austausch des Dateisystems um den Zugriff zu abstrahieren. Es stehen ldiskfs und ZFS zur Verfügung. Bei ldiskfs handelt es sich um ein optimiertes ext4 Dateisystem und ermöglicht, dass 4 Milliarden Dateien pro MDT gespeichert werden können. Seit Lustre 2.4 steht auch das modernere ZFS zur Verfügung. Dies ermöglicht größere Dateien (2^63 Bytes), mehr Dateien pro MDT (256 Milliarden) und verfügt über weitere komfortable Features wie Snapshots und Prüfsummen. Zusätzlich ist die Konfiguration einfacher und erfordert keine Anpassungen des Kernels, wie es bei ldiskfs der Fall ist.

2.3 File Striping

Um eine hohe Performance zu gewährleisten, adaptiert Lustre Mechanismen, die ähnlich bei einem RAID 0 System zu finden sind. Dateien werden in Blöcke aufgeteilt, die im Fall von Lustre auf unterschiedlichen Servern gespeichert werden. Angenommen eine

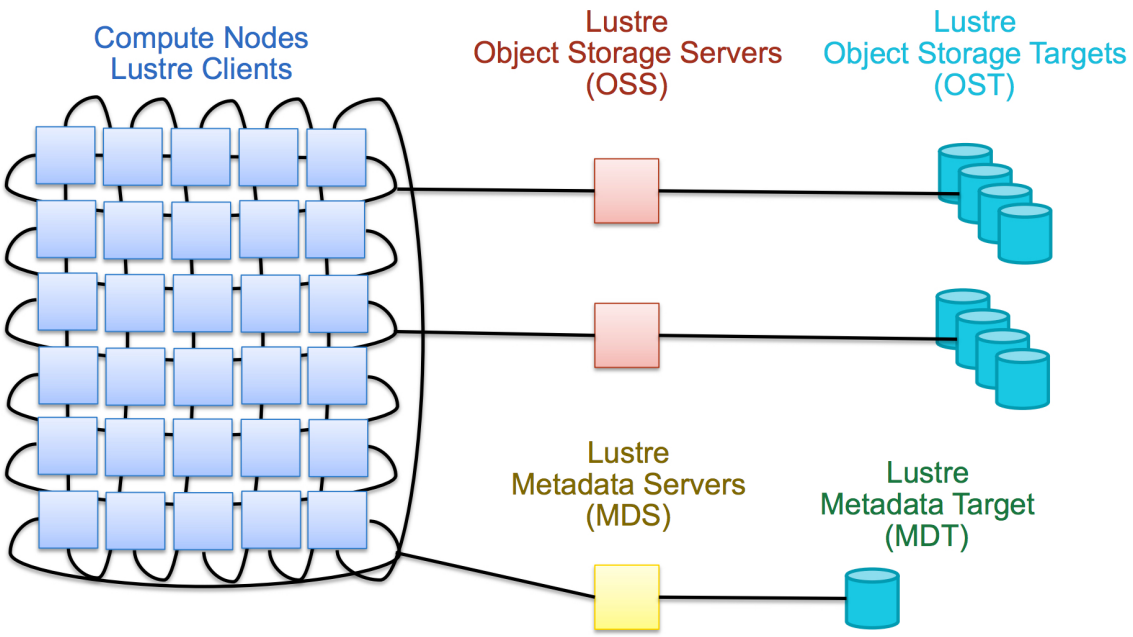

Figure 2.1: Architektur [htt16]

Datei wird in 2 Blöcke aufgeteilt und auch auf 2 Servern gespeichert. Ein paralleler Zugriff auf diese beiden Blöcke ermöglicht, dass die Ressourcen der beiden Server optimal und unabhängig von einander zum abrufen der Datei genutzt werden kann.

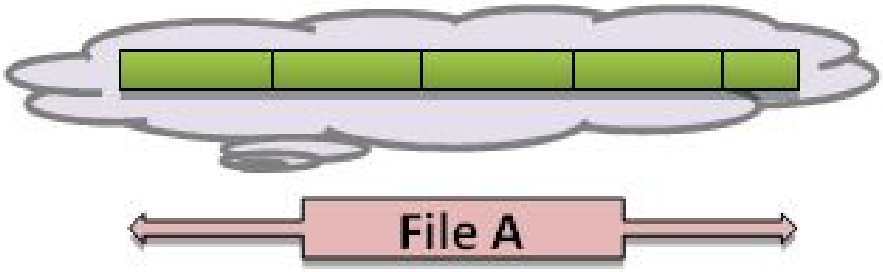

Figure 2.2: Dateisicht [cit16]

File Striping ermöglicht das Aufteilen einer Datei auf mehrere OSTs. Für die Effektivität dieser Verteilung sind 2 Parameter maßgeblich. Der Parameter Strip Count bestimmt auf wie vielen OSTs die Datei verteilt werden soll. Stripe Size bestimmt die Größe (in Bytes) der einzelnen Stripes. Diese Konfiguration kann für einzelne Dateien, Ordner und ganze Dateisysteme festgelegt werden. In der Regel wird dabei nicht festgelegt auf welchen OSTs die Stripes abgelegt werden. Um die Server möglichst gleichmäßig zu belegen werden die OSTs durch einen Round Robin oder gewichtete Algorithmen bestimmt.
Die richtige Wahl dieser beiden Parameter ist essentiell für die Leistung des Dateisystems.

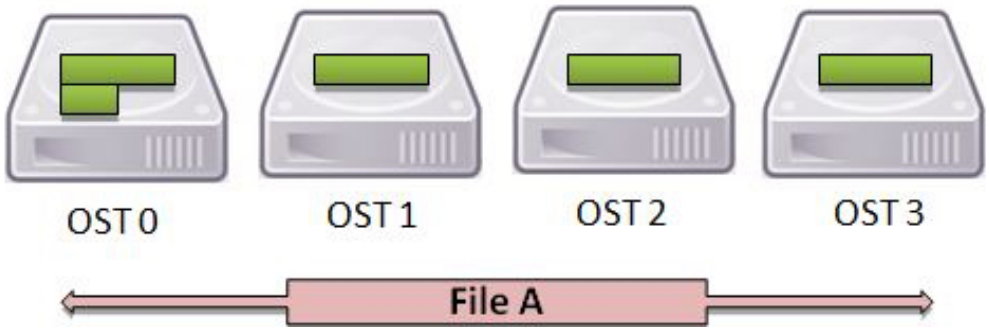

Figure 2.3: physische Sicht

Oftmals sind Testläufe mit verschiedenen Parametern angebracht. Folgende Regeln sollen Helfen diese Werte besser zu bestimmen.

2.3.1 gleichmäßige Ausrichtung

Im Idealfall erreicht man eine gleichmäßige Ausrichtung der Stripes. In folgendem Beispiel wird von einer Datei mit der Größe von 9 MB ausgegangen. Der Stripe Count beträgt 4, während der Stripe Size 1 MB ist. Das ermöglicht eine gleichmäßige Verteilung der Stripes mit einer jeweiligen Größe von 1 MB. Ein einzelner Prozess muss nur auf einem OST arbeiten.

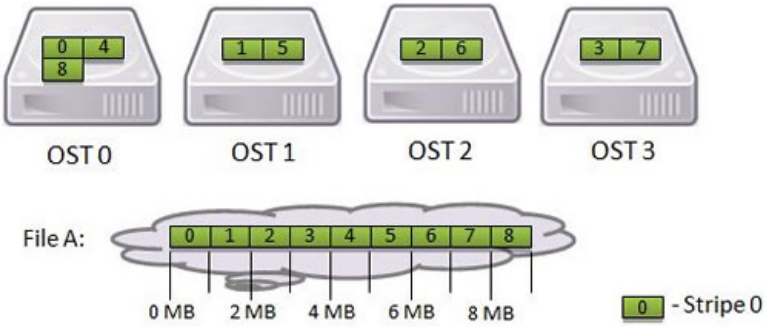

Figure 2.4: Aligned Stripes

2.3.2 ungleichmäßige Ausrichtung

Das folgende Beispiel veranschaulicht, was passieren kann wenn die Parameter falsch gewählt sind. Angenommen, der Stripe Count beträgt 4, was auch der Anzahl der beteiligten Prozesse entspricht. Die Datei ist 5 MB groß und der Stripe Size beträgt 1 MB. Dies führt zu einer starken Fragmentierung, da dies zwangsläufig dazu führen wird, dass ein Prozess auf verschiedenen OSTs arbeiten muss. Das führt zu Performance Einbußen, da mehrere Anfragen gestellt werden müssen (s. Figure 2.5).

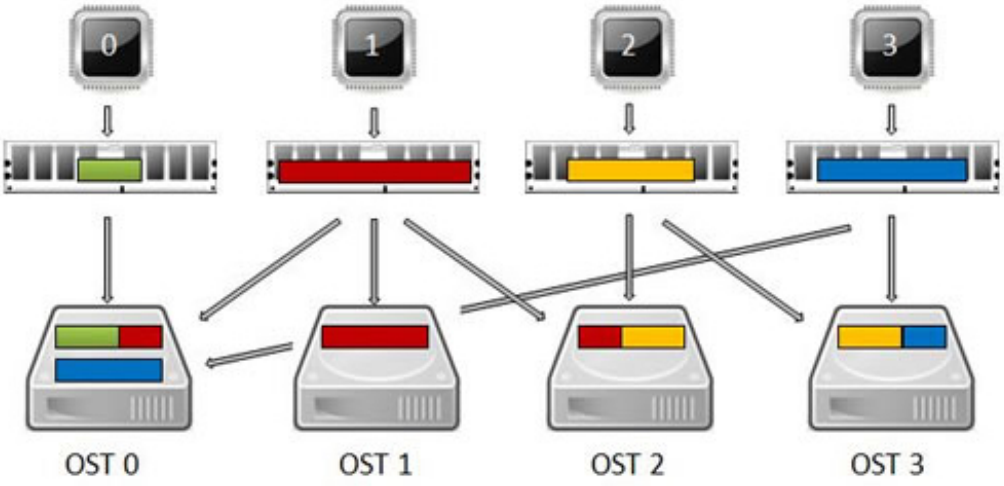

Figure 2.5: Non-Aligned Stripes

2.3.3 Bestimmung der Parameter

Die richtige Wahl der Parameter hängt von verschiedenen, kontextbezogenen Faktoren ab.

Stripe Size

Der maximale Stripe Size beträgt 4 GB. Diese Wahl ist besonders dann sinnvoll, wenn eine sehr große Datei abgerufen werden soll, da weniger Locks verwaltet werden müssen. Ein einzelner Prozess behält so länger das exklusive Schreib/Leserecht, ohne das dies neu angefordert werden müsste. In den meisten Fällen bietet sich jedoch ein Stripe Size von 1 MB bis 4 MB an. Da das Lustre Dateisystem Chunks von 1 MB Größe verschickt sind kleiner Werte unter Umständen kontraproduktiv für die Performance. Des weiteren muss der Stripe Size, aus Gründen der Abwärtskompatibilität, ein Vielfaches von 64 KB sein (Speicherseite in ia64 Prozessoren).

Stripe Count

Bei kleinen Dateien ist es sinnvoll einen Stripe Count von 1 zu wählen. Andernfalls würde die Datei in viele Stripes aufgeteilt werden, was dazu führen würde, dass mehrere Anfragen zum Abrufen einer Datei nötig sind [NAS16]. Ein großer Wert für den Stripe Count hilft, wenn der Zugriff auf eine Datei durch die zur Verfügung stehende Bandbreite eines einzelnen Servers begrenzt wird. Dieses Szenario ist denkbar, wenn beim Start einer parallelen Anwendung die gleichen Daten als Ausgangspunkt für die Prozesse dienen, oder wenn viele Daten gespeichert werden müssen.

2.4 File Identifiers (FIDs)

FIDs repräsentieren Geräte-unabhängig die klassischen Inode Kennziffern. Jede FID ist 128-Bit lang und enthält 3 Eigenschaften. Die einzigartige Sequence Number (64-Bit) dient zur Identifizierung des MDTs. Die Object ID (32-Bit) identifiziert das jeweilige

Objekt, während die Version Number (32-Bit) eine Versionierung von Dateien erlaubt (derzeit noch nicht genutzt) [Cow15].

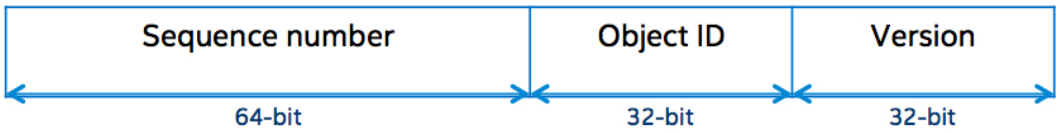

Figure 2.6: File Identifier

Um eine Datei zu öffnen wird die FID genutzt um bei dem MDT das File Layout abzurufen. Dieses Layout beinhaltet die OSTs auf denen die benötigten Stripes enthalten sind.

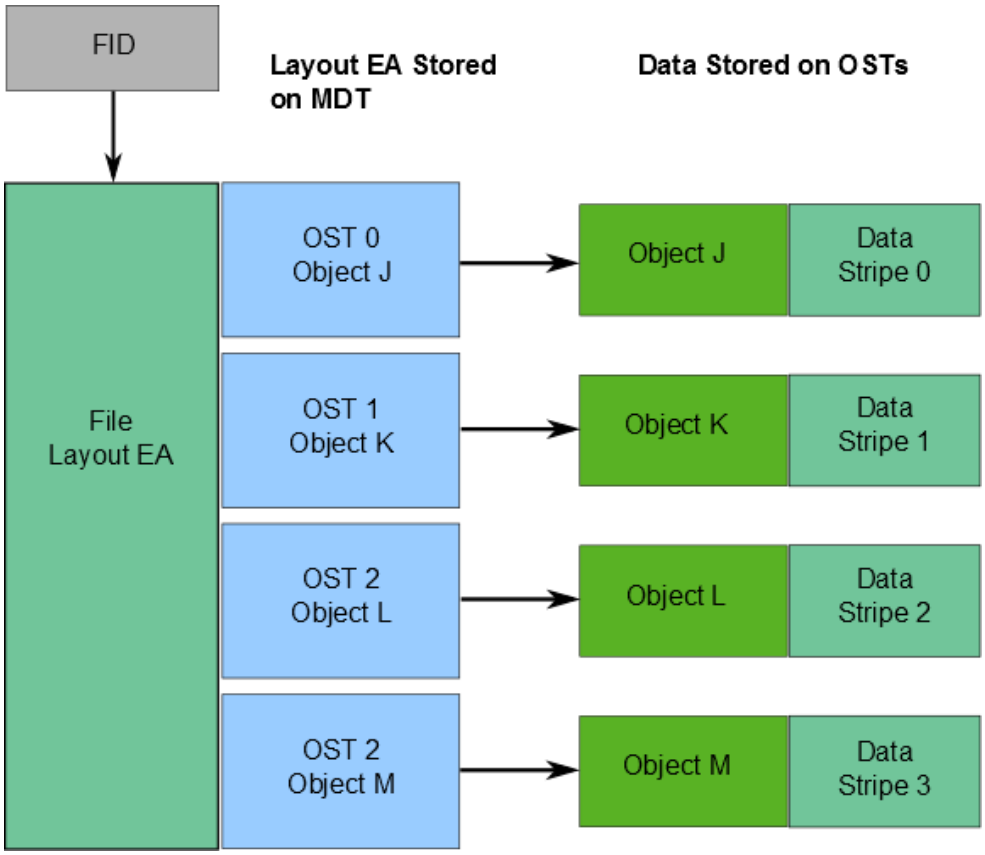

Figure 2.7: File Layout[Lus16]

2.5 Locking

Das Locking ist für Lustre unvermeidbar, da viele Prozesse gleichzeitig auf einem gemeinsamen Datenbestand arbeiten. Der Lustre distributed lock manager (LDLM) sichert die Kohärenz zwischen allen Servern und Clients. Locking findet auf 2 Ebenen

statt. Die Metadata Locks (MDS LDLM) organisieren Modifikationen an Inodes auf den MDTs. Des weiteren werden auf den OSTs die Locks für die Stripes verwaltet. Bei einem exklusiven Lock kann nur der Prozess/Client Veränderungen vornehmen, der diesen Lock angefordert hat. Wenn mehrere Clients den gleichen Abschnitt anfordern entstehen also Wartezeiten, wodurch die Performance sinkt. Read Locks sind dagegen unproblematisch und ermöglichen eine starke Parallelisierung.

2.6 Lustre Network (LNET)

Die von Lustre benötigte Netzwerkinfrastruktur wird durch LNET implementiert. Die dafür nötigen Treiber werden durch den Lustre Network Driver (LND) bereitgestellt. Zu den unterstützten Netzwerktypen zählen unteranderem TCP (10 GigE, IPoIB), Infiniband und Cray. Das LNET wird sowohl zur Datenübertragung, als auch für die Metadaten und Locks genutzt.

2.7 RDMA

Remote Direct Memory Access hat den Vorteil, dass die Network Interface Card (NIC) direkt in den RAM des Ziels schreiben kann. Dies bedeutet, dass Kommunikation ohne Involvierung der CPU oder der Caches möglich ist. Echte RDMA ist jedoch nur bei Infiniband Übertragungen möglich.

3 Hochleistungsrechnen

3.1 Failover

Besonders beim Hochleistungsrechnen, dem hauptsächlichen Einsatzgebiet von Lustre, ist die Vermeidung von Downtime sehr wichtig. Um dies zu bewerkstelligen sind verschiedene Failover Szenarien möglich. In der Active/Passiv Kombination übernimmt ein passiver Server die Aufgaben des Aktiven. Zu jeder Zeit ist also die benötigte Hardware doppelt vorhanden. Bei der Active/Active Konstellation kann jeder Server beim Ausfall des Anderen dessen Aufgaben übernehmen. In dieser Zeit kann der Defekt des ausgefallenen Servers behoben werden. Sobald die Probleme behoben wurden, werden die Aufgaben wieder gleichmäßig aufgeteilt. Die Fehlererkennung beschränkt sich bei Lustre auf das Dateisystem. Durch externe Mechanismen wie Corosync oder Pacemaker kann dies jedoch auf die Hardwareebene erweitert werden.

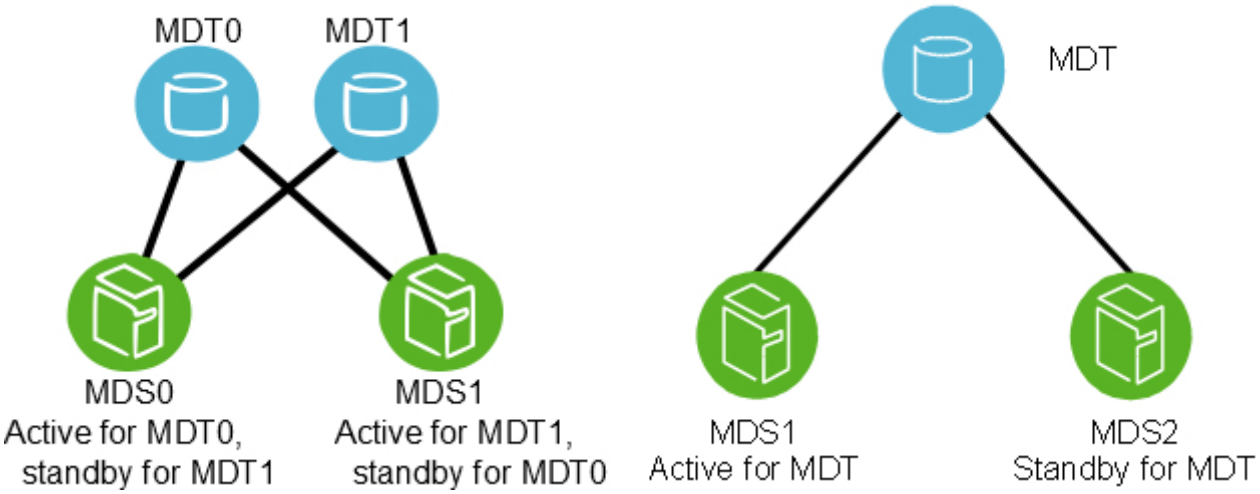

Figure 3.1: Failover

3.2 Quotas

Quotas ermöglicht den Speicherplatz der Nutzer zu beschränken. Diese Beschränkungen können sowohl für die Anzahl der Inodes (MDT) gelten, als auch für die Anzahl der Blöcke (OSTs)[Int15]. Dabei wird zwischen dem Softlimit und dem Hardlimit unterschieden. Hardlimit fungieren als eine obere Schranke, die durch den Nutzer nicht überschritten

werden kann. Das Softlimit darf für eine bestimmte Zeit, der Grace Period, überschritten werden. Anschließend verhält sich das Softlimit wie das Hardlimit.

3.3 Hierarchical Storage Management (HSM)

Als HSM Storage werden langsamere Tape Archive bezeichnet. Sie sind günstiger als herkömmliche Serverfestplatten und haben eine längere Haltbarkeit. Das schnelle Speichersystem, welches durch Lustre verwaltet wird, dient als Cache für Daten auf die oft zugegriffen wird. Gleichzeitig besteht eine schnelle Anbindung an das Archiv, sodass bei Zugriffen auf inaktive Daten der Reaktivierungsprozess möglichst schnell geschieht. Dieser Archivierungs- und Wiederherstellungsprozess wird von einem Client, dem Agenten, koordiniert. Dieser Coordinator muss auf jedem MDT aktiviert sein.

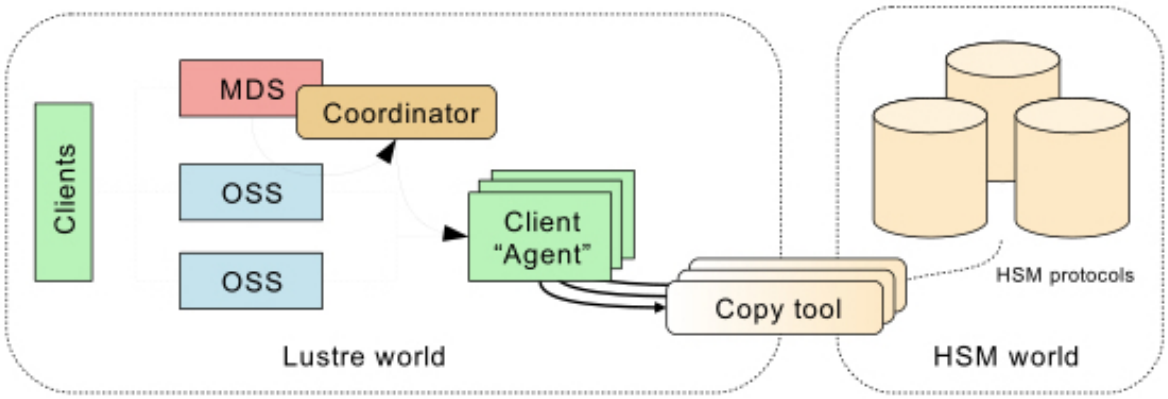

Figure 3.2: HSM

3.4 Zugriff

Auf jedem Server, auf dem der Lustre Client installiert wurde, kann durch den bekannten mount Befehl eine Verbindung zu Lustre hergestellt werden.

```
1 mkdir -p /mnt/lustre/client
2 mount -t lustre $(MDS):/lustre /mnt/lustre/client
```

Listing 3.1: Lustre Mount

Zusätzlich ermöglich Lustre die Einbindung unter Windows und OS X durch NFS und CIFS Schnittstellen. Die parallele Programmierung erfolgt durch die llapi oder MPI-IO Implementierung.

4 Performance Vergleich

Im Folgenden werden verschiedene Zugriffsmöglichkeiten vorgestellt und verglichen. Die Leistung wurde auf dem Cray XT5 (Jaguar) gemessen.

4.1 Single Writer I/O

In folgendem Fall senden die Prozesse ihre Schreib- und Leseanforderungen an einen einzelnen Prozess. Dieser ausgewählte Prozess ist für die gesamte Kommunikation mit Lustre zuständig.

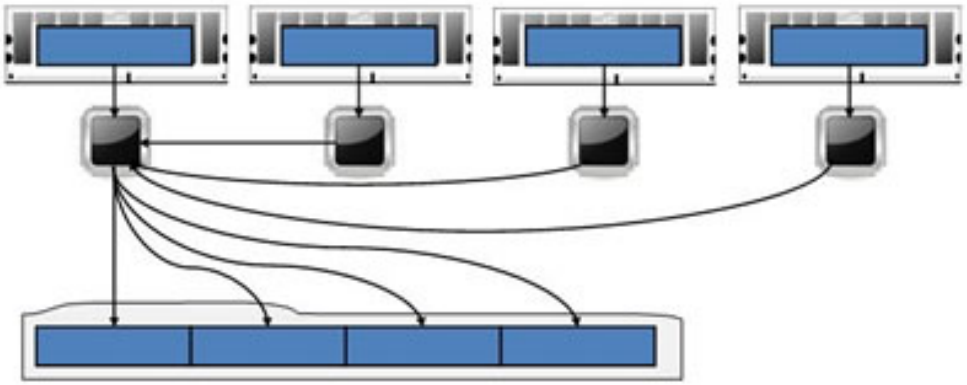

Figure 4.1: Single Writer I/O

Der folgende Test wurde mit einer Dateigröße von 32 MB bis 5 GB gemacht, um die Last je OST gleichmäßig bei 32 MB zu halten. Die Stripe Size Parameter sind 1 MB und 32 MB groß.

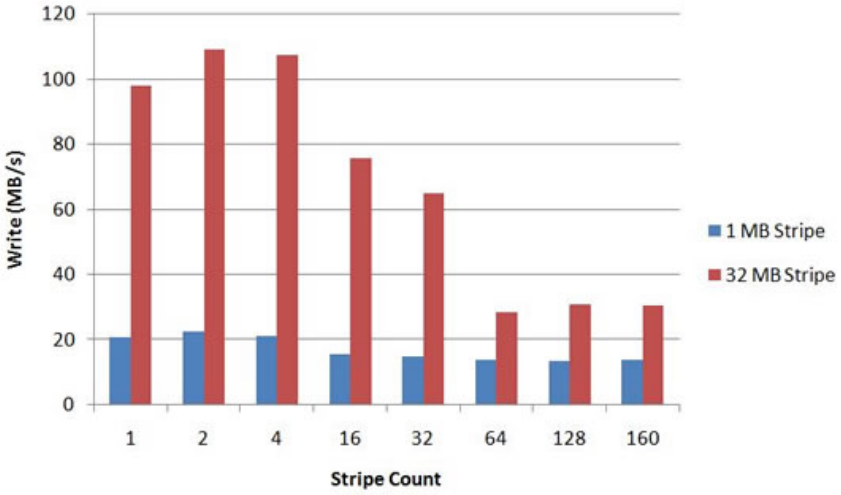

Figure 4.2: Single Writer I/O

Dieser Test zeigt sehr deutlich wie wichtig es ist, dass ein Prozess mit möglichst wenigen OSTs kommunizieren muss. Dies erklärt, weshalb eine so hohe Anzahl von OSTs die Performance so drastisch verringert. Die höchste Geschwindigkeit wird mit einem Stripe Size von 32 MB erreicht. Zu kleine Stripes führen zu einer größeren Netzwerkbelastung, da jeweils neue Anfragen gestellt werden müssen.

4.2 Single Shared File I/O

In diesem Beispiel können die Prozesse selbstständig und parallel kommunizieren.

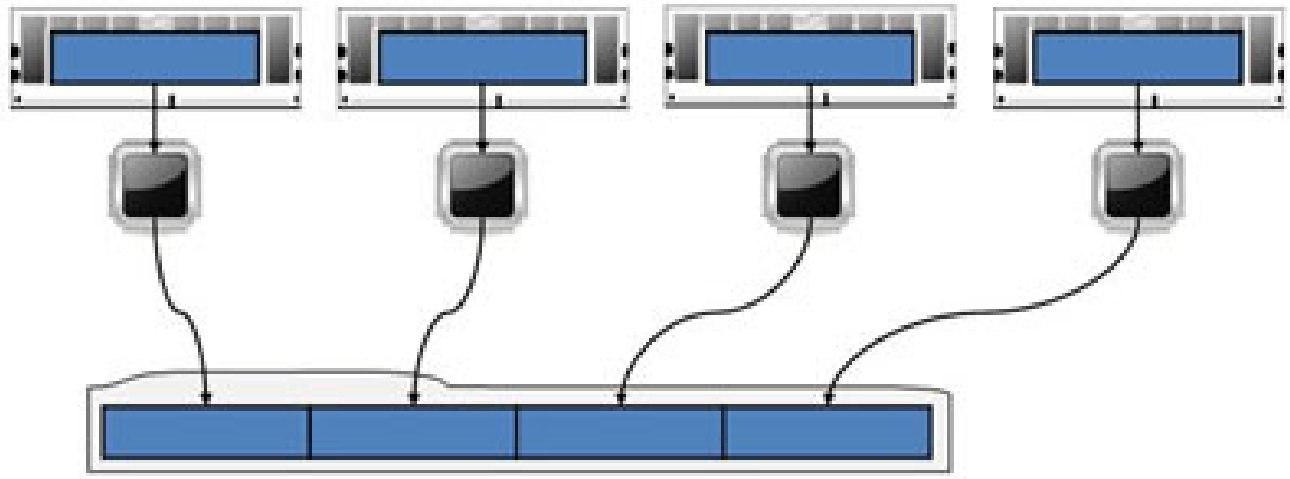

Figure 4.3: Single Shared I/O

Es werden 2 verschiedene Zugriffscharakteristika verwendet. In dem Layout #1 werden feste Blöcke zugeteilt, die sich während der weiteren Laufzeit nicht mehr ändern. Die

Blöcke sind dabei 32 MB und 64 MB groß. Layout #2 spricht immer Blöcke der Größe 1 MB oder 2 MB an. Nach Bearbeitung werden dem Prozess neue Abschnitte zugewiesen.

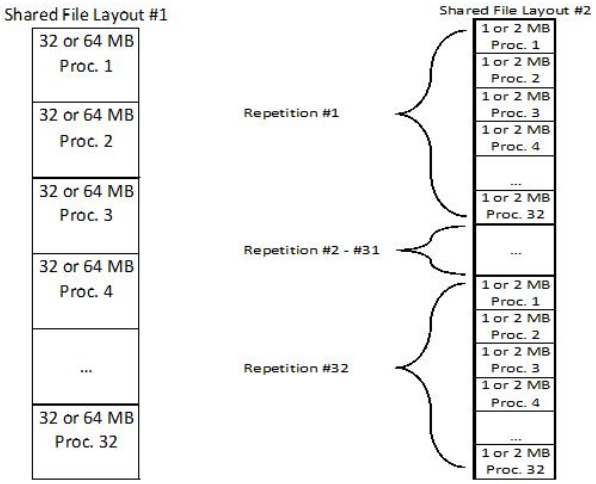

Figure 4.4: Shared Layout

Nun wird schreibend auf eine Datei mit der Größe 1 GB (32 OSTs) und 2 GB (64 OSTs) zugegriffen.

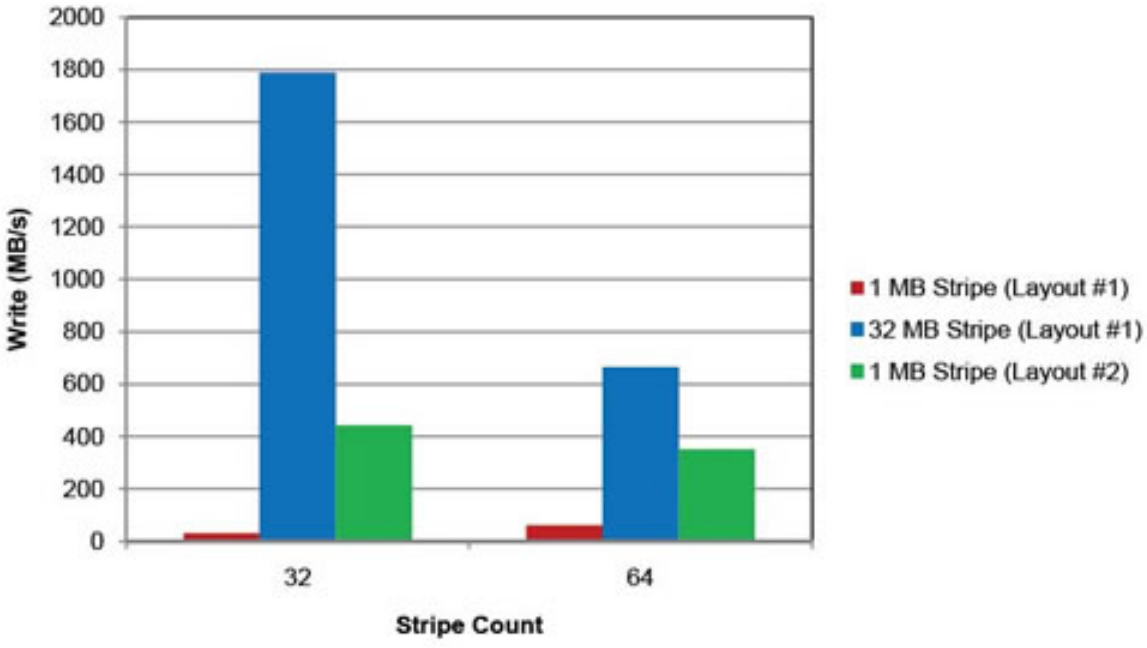

Figure 4.5: Single Shared

Die schlechteste Performance wird in Verbindung mit dem Layout #1 und einem Stripe Size von 1 MB erreicht. Jeder zugeteilte 32 MB Block muss in 1 MB Blöcke geteilt

werden und es muss jeweils ein anderer OST angesprochen werden. Die mit Abstand beste Schreibgeschwindigkeit wird bei Verwendung des Layouts #1 in Verbindung mit einem Stripe Size von 32 MB gemessen. Es sind Geschwindigkeiten von bis zu 1800 MB/s möglich. Das lieg dadran, dass die Prozesse nicht die OSTs wechseln müssen, sondern ihren vollständigen Schreibvorgang mit einer Verbindung erledigen können. Das Layout #2 mit einem Stripe Size von 1 ist deshalb schneller als in Layout #1, da hier jeder Prozess nur auf einem einzigen OST arbeiten muss. Aus dem gleichen Grund sinkt die Performance bei Stripe Count 64, da die Prozesse zwangsläufig auf mehrere OSTs zugreifen müssen.

4.3 File Per Process I/O

In diesem Fall hat jeder Prozess Zugriff auf eine vollständig unabhängige Datei. Dies ermöglicht maximale Parallelität und Performance, da z.B. keine Locks verwaltet werden müssen.

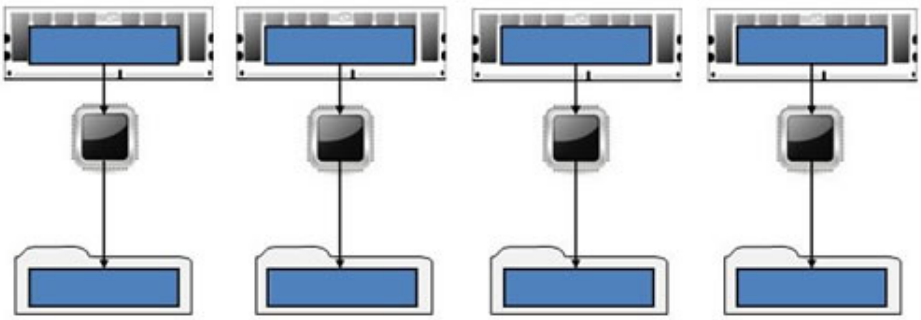

Figure 4.6: File Per Process I/O

In diesem Test ist der Stripe Count 1 und der Stripe Size beträgt 1 MB und 32 MB. Bei einem Stripe Size von 1 MB wird die höchste maximal Geschwindigkeit erreicht. Wenn die Anzahl der Prozesse jedoch weiter steigt, verschlechtert sich die Geschwindigkeit sehr schnell. Das liegt dadran, dass viele einzelne Datei verwaltet werden müssen. Die Wahl des Stripe Size von 32 MB Größe ist also zu empfehlen, da die Performance kalkulierbar und gleichmäßig ist.

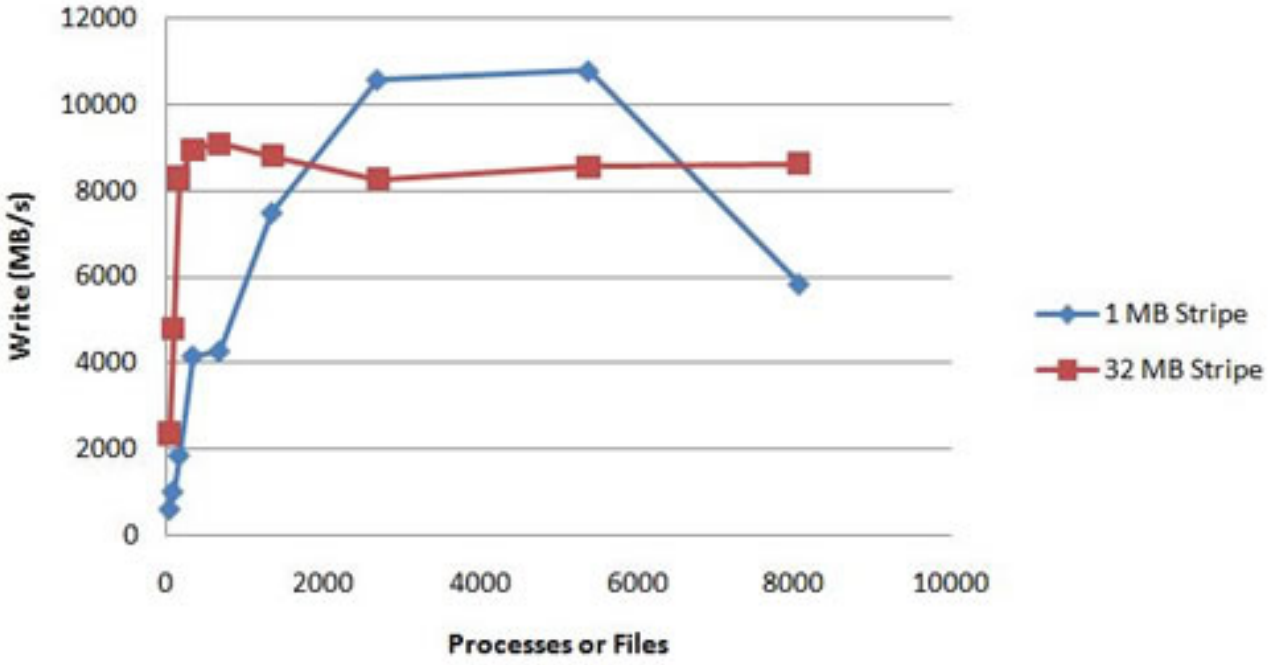

Figure 4.7: File Per Process I/O

5 Conclusion

Zusammenfassend lässt sich festhalten, dass Lustre durch den parallelen Datenzugriff die Bandbreite eines Systems deutlich steigert. Die dafür nötigen Parameter müssen sorgsam festgelegt werden und hängen stark von den Bedürfnissen der Anwendung ab. Die komplexe Einarbeitung und die vielen Funktionen für den Hochleistungsbetrieb prädestinieren Lustre für den Einsatz in Rechenzentren mit großen Speichersystemen.

Bibliography

[cit16] citutor. Dateisicht. Mai 2016. https://www.citutor.org/.

[Cow15] Malcolm Cowe. AN INTRODUCTION TO LUSTRE ARCHITECTURE. August 2015. https://nci.org.au/wp-content/uploads/2015/08/01-Introduction-to-Lustre-Architecture.pdf.

[htt16] https://www.ornl.gov/. Lustre Architektur. Mai 2016. http://lustre.ornl.gov/lustre101-courses/content/C1/L1/LustreIntro.pdf.

[Int15] Intel. Lustre system and network administration - Lustre quota. Juli 2015. https://nci.org.au/wp-content/uploads/2015/08/04-Quota.pdf.

[Lus16] Lustre. Lustre* Software Release 2.x. Mai 2016. http://doc.lustre.org/lustre_manual.xhtml.

[NAS16] NASA. Lustre Best Practices. Mai 2016. http://www.nas.nasa.gov/hecc/support/kb/lustre-best-practices_226.html.

[SAS16] Serial Attached SCSI. Mai 2016. https://de.wikipedia.org/wiki/Serial_Attached_SCSI.

Appendices

List of Figures

List of Listings