Diplomarbeit

Philipp Gruber

Implementierung eines verteilten
I/O-Scheduling-Mechanismus für Storages Area Networks
unter Linux

Hochschule Niederrhein
Fachbereich Elektrotechnik und Informatik
Studiengang Technische Informatik

Betreuer:
Prof. Dr. Jürgen Quade
Dipl.-Ing. Lars Fürst

21. Januar 2008

Inhaltsverzeichnis

1 Einführung

Dieses Kapitel erläutert den speziellen Anwendungsfall der digitalen Filmproduktion auf dessen Gebiet die Motivation dieser Diplomarbeit begründet ist.

1.1 Einleitung

Im Rahmen dieser Diplomarbeit liegt der Fokus auf dem konkreten Anwendungsfall der digitalen Filmproduktion.

In der professionellen Filmproduktion gewinnt die digitale Technik immer mehr an Bedeutung. Während die Nachbearbeitung von Kinofilmen heutzutage bereits grundsätzlich in digitaler Form stattfindet, werden in vielen Filmstudios aus Kostengründen noch analoge Kameras eingesetzt. Doch auch hier ist ein stetiger Anstieg des digitalen Anteils festzustellen.

Die Vorteile der digitalen Filmproduktion sind vielfältigere Möglichkeiten im Bereich Spezialeffekte, sowie die Möglichkeit der verlustfreien Replikation und Übertragung. Durch hohe Ansprüche an Qualität, hauptsächlich in Kinos, aber auch für die sich inzwischen durchgesetzt habenden Formate wie HD-TV entstehen ebenfalls höchste Ansprüche an die zur Verarbeitung verwendete Hardware. Die für einen 90-minütigen Kinofilm anfallenden Datenmengen können eine Gesamtmenge von 100 Terabyte überschreiten. Entsprechend sind die Anforderungen an Größe und Performance der Datenhaltung.

Um diese Daten zu verwalten werden Speichernetze eingesetzt, die es ermöglichen von vielen Workstations über hochperformante Verbindungen auf zentrale Datenspeicher in Form von Festplattenverbünden zuzugreifen.

Doch auch diese stoßen bei der Verwaltung von mehreren parallelen Zugriffen schnell an ihre Grenzen. Analysen der Thomson Corporate Research haben jedoch ergeben, dass die in der Filmproduktion anfallenden Datenströme besonders günstige Eigenschaften vorweisen, die es ermöglichen durch Koordination der Workstations die Performance erheblich zu optimieren.

Dazu wurde in einer vorangegangenen Diplomarbeit [Fü07] das Konzept des verteilten I/O-Schedulings entworfen, welches nun auf die speziellen Bedürfnisse einer digitalen Produktionsstätte zugeschnitten werden soll.

1.1.1 Gliederung

- Im weiteren Verlauf des ersten Kapitels wird detailliert auf den Anwendungsfall eingegangen.
- Das zweite Kapitel beschreibt die technischen Grundlagen, welche für das Verständnis der Arbeit notwendig sind.
- Im dritten Kapitel wird auf die Problematik eingegangen werden, die im Anwendungsfall besteht.
- Kapitel vier beschreibt die theoretischen Ansätze, mit denen die bestehenden Probleme gelöst werden können.
- In Kapitel fünf wird die Umsetzung in Form der Implementierung vorgestellt und welche technischen Besonderheiten es zu beachten gab.
- Die fertige Implementierung wird in Kapitel sechs getestet und damit die Problemlösungen verifiziert.
- Anschließend wird ein Fazit gezogen und ein Ausblick auf mögliche zukünftige Weiterentwicklungen in dem Bereich gegeben.

1.2 Anwendungsfall der digitalen Filmproduktion

Die Filmproduktion entwickelt sich heutzutage immer stärker von analogen zu digitalen Techniken hin. Anstelle der immer noch weit verbreiteten analogen Zelluloid-Bänder werden zunehmend digitale Speicher eingesetzt. In der Postproduktion haben sich diese inzwischen vollständig durchgesetzt. Da die bei der Filmproduktion anfallenden Daten viele Terabyte umfassen, sind Festplatten die einzige Möglichkeit diese digital mit hoher Datenrate abzuspeichern.

Das gedrehte Filmmaterial wird entweder direkt von einer Kamera oder später per Filmscanner von einer Filmrolle auf dem Datenspeicher abgelegt. Zur Weiterverarbeitung muss es auch wieder abgespielt oder auf andere Datenträger kopiert werden.

Für aktuelle Kinofilme besteht ein hoher Anspruch an Qualität. Standards wie HD, 2K und 4K sind heutzutage für moderne Filme unabkömmlich. Bei der Akquirierung und bei vielen Produktionsschritten wird mit Auflösungen von bis zu 4096*3112 Pixeln gearbeitet, was dem Standard 4K entspricht.

Um bei der Produktion möglichst hohe Performance und keine Qualitätsverluste durch Kompressionsverfahren wie MPEG-4 zu erreichen, wird ausschließlich mit unkomprimiertem Bild- und Tonmaterial gearbeitet.

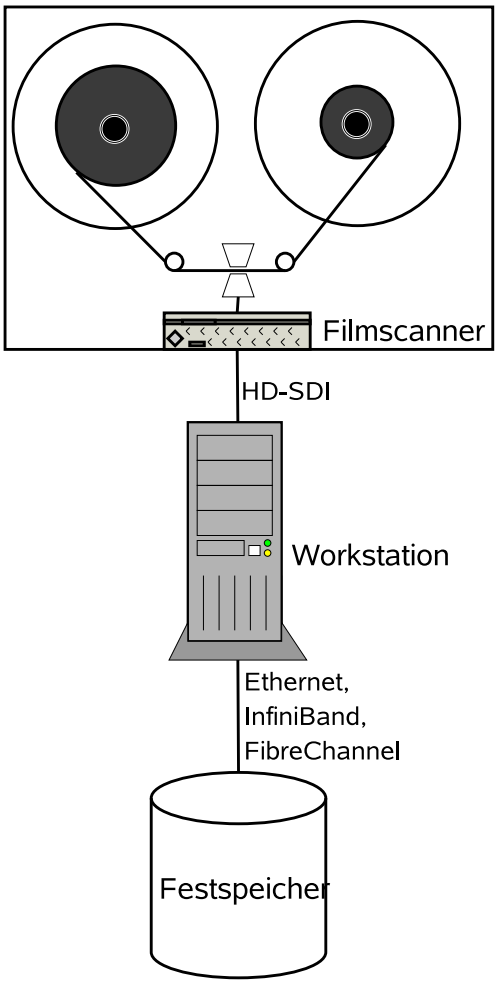

Abb. 1: *Datenfluss von einem Filmscanner auf einen Datenspeicher*

Format	Auflösung	Datenrate
HD	1920*1080	197 MB/s
2K	2048*1556	291 MB/s
4K	4096*3112	1167 MB/s

Tab. 1: *Datenraten verschiedener Videostandards bei 10 Bit Farbtiefe. Quelle: [Dis04], Seite 4*

Bei einer Bildrate von 24 Bildern pro Sekunde und einer Farbtiefe von 10-16 Bit ergeben sich hierbei Datenraten von bis zu 1,2 GB/s (siehe Tab. 1).

Für das Überspielen von Filmmaterial von einer Kamera auf einen Datenspeicher sind Übertragungstechniken wie HD-SDI (High Definition Serial Digital Interface) üblich. Solche Techniken übertragen sämtliche Daten in Echtzeit, das heißt mit einer konstanten Datenrate. Der Scanvorgang, also das Konvertieren von einem analogen Filmband in ein digitales Format ist ein Prozess, der ebenfalls eine konstante Datenrate liefert, da der Scanvorgang mit gleichbleibender Geschwindigkeit abläuft.

1.3 Systembeschreibung

Eine typische digitale Produktionsstätte besteht aus einem oder mehreren großen Datenspeichereinheiten, mehreren Produktionsrechnern, für Aufgaben wie rendern, schneiden oder das Anwenden von Effekten. Außerdem gibt es bei Verwendung von analogen Kameras Filmscanner welche die Bänder digitalisieren. Digitale Kameras können die Daten direkt digital ausspielen. Nach der Bearbeitung werden die Daten mittels eines Filmprinters auf analoge Filmrollen gedruckt.

Die meisten Geräte sind nicht direkt an den Datenspeicher angeschlossen, da die Übertragungsverfahren nicht kompatibel sind. Stattdessen sind sie mit einer Workstation verbunden die wieder direkt am Datenspeicher angeschlossen ist. Diese speichert die Bilddaten, die beispielsweise per HD-SDI oder einem anderen synchronen Protokoll eingelesen werden auf einem Datenträger ab.

Eine solche Aufgabe kann ein leistungsfähiger Computer auch parallel für mehrere Geräte übernehmen.

Während zum Beispiel zum Anwenden von Effekten eine hohe Lese- und Schreibgeschwindigkeit zwar wünschenswert aber nicht notwendig ist, ist diese für Geräte wie Filmscanner oder Filmprinter unbedingt notwendig. Deshalb ist es wichtig, dass die Datensenke die Daten schnell genug annehmen und abspeichern kann.

Beim Abspeichern des Materials würden bei zu geringer Datenrate Einzelbilder - auch Frames genannt - verloren gehen und das Material wäre unbrauchbar.

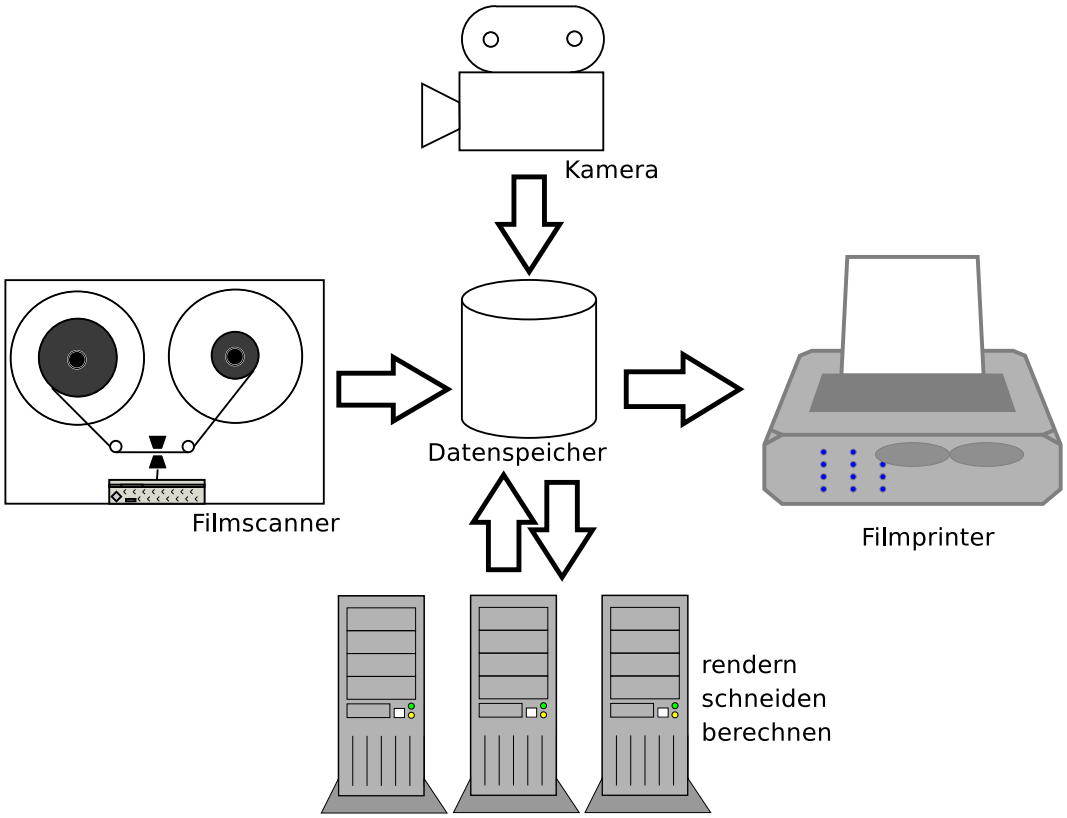

Abb. 2: *Schema einer Postproduktionumgebung*

Der gleiche Effekt kann beim ausbelichten auftreten. Wenn dort Frames verloren gehen, fehlen diese auf der Filmrolle.

Die benötigte Datenrate muss also für die Applikation unbedingt kontinuierlich zur Verfügung stehen.

2 Grundlagen

In diesem Kapitel werden die technischen Grundlagen, welche zum Verständnis der Arbeit notwendig sind, sowie der aktuelle Stand der Technik erläutert

2.1 Paralleler Zugriff auf Datenspeicher

Oft ist es notwendig, dass von verschiedenen Arbeitsrechnern aus zeitgleich auf einen Datenspeicher zugegriffen werden kann. Dazu existieren verschiedene Ansätze.

Ein *Network Attached Storage* ist ein Gerät oder ein Rechner der ein lokales Dateisystem über ein Standard-IT-Netzwerk wie Ethernet anderen Rechnern zur Verfügung stellt. Dies geschieht in der Regel über ein dateiorientiertes Protokoll wie NFS, SMB oder CIFS. Das bedeutet, ein Client kann direkt auf Dateien des Servers zugreifen. Anstelle eines Dateisystemtreibers braucht er lediglich einen Client für das Protokoll.

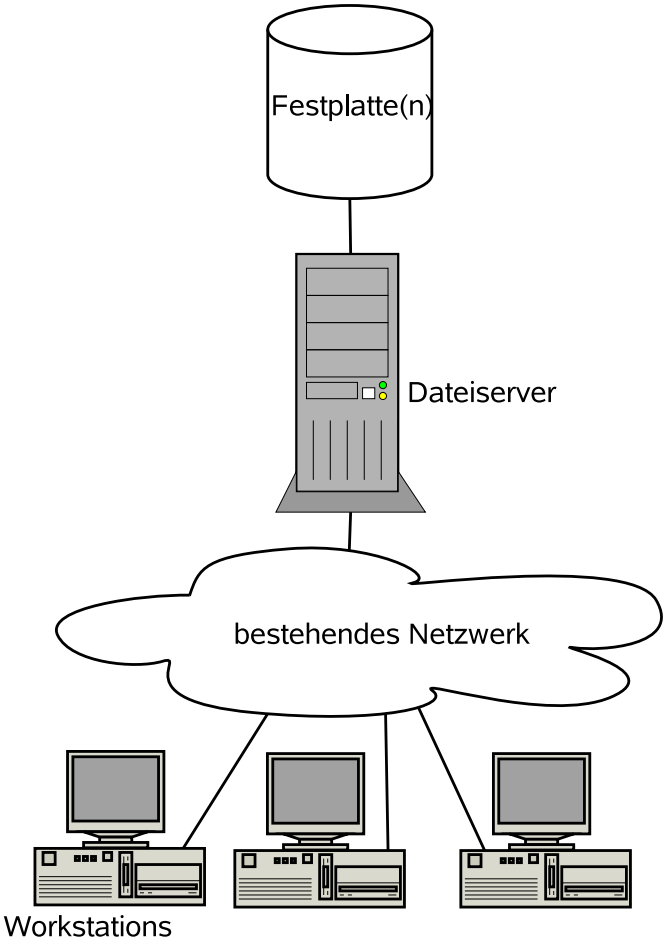

Abb. 3: *Beispiel eines Dateiservers mit 3 Workstations*

Teilweise sind NAS vorkonfiguriert und bestehen

aus spezialisierten oder abgespeckten Komponenten und Betriebssystemen (siehe [TE03], S. 234).

Ein *Storage Area Network* (SAN) stellt den Datenspeicher den Clients direkt blockorientiert zur Verfügung. Das heißt, ein Client kann einzelne Blöcke des Speichers anfordern oder beschreiben. Eine Verwaltungseinheit wie ein Dateiserver oder die Steuerungseinheit eines NAS existiert im Datenpfad nicht. Die Festplattenverbünde verfügen lediglich über einen Storage-Controller welcher sie zu einem logischen Laufwerk zusammenfasst. Die Clients werden über Netzwerk direkt an diesen Storage-Controller angeschlossen.

Je nach Anwendungsgebiet hat jedes dieser Systeme Vor- und Nachteile. Ein NAS ist leicht zu installieren und administrieren und außerdem kostengünstig. Je nach Bedarf kann ein leistungsfähiger Server eingesetzt werden, der viele Zugriffe parallel verwalten kann.

Ein NAS arbeitet mit Standardhardware und -protokollen wie Ethernet und TCP/IP, weshalb es leicht in bestehende Infrastrukturen eingebunden werden kann.

Ein SAN ermöglicht durch direkten blockorientieren Zugriff ohne verwaltende Instanzen eine höchstmögliche Performance. Einsatzort für ein SAN sind Umgebungen die höchste Anforderung an Bandbreite haben. Bestehende Netzwerkstrukturen wie Ethernet sind hier nicht ausreichend; Es werden dedizierte Netzwerkverbindungen auf Basis von Techniken wie iSCSI, InfiniBand und FibreChannel genutzt. Die Bandbreite kann noch erhöht werden, in dem mehrere Leitungen parallel eingesetzt werden. Diese Technik wird *Multipathing* genannt (Siehe [TE03], Seite 343). Da mehrere redundante I/O-Pfade bestehen wird dadurch ebenfalls die Ausfallsicherheit erhöht.

2.2 Physikalische Eigenschaften von Festplatten

Festplatten sind magnetische Datenspeicher mit hoher Kapazität. Die Daten werden auf mehreren Scheiben angelegt und von Lese- oder Schreibköpfen verarbeitet, welche jeweils in die richtige Spur der entsprechenden Datenscheibe springen. Je verteilter die Daten gespeichert sind, desto häufiger muss der Kopf zwischen den Spuren hin- und herspringen. Diese Sprünge gehören zu

den langsamsten Operationen eines modernen Computers (Siehe [Lov05], Seite 243). Hinzu kommt, dass der Kopf danach warten muss, bis die Scheibe sich soweit gedreht hat, dass der Kopf an der richtigen Stelle steht.

Bei einer Umlaufgeschwindigkeit 7.200 Umdrehungen pro Minute dauert eine volle Umdrehung

$$U = \frac{1}{7200\frac{U}{Min}} = 8,\overline{3}ms$$

Der Sprung der Köpfe in eine andere Spur kann bis zu 23 ms dauern (Siehe [Son06], Seite 56).

Im optimalen Fall wird sequentiell auf die Festplatte zugegriffen, das heißt dass jeder einzelne Request da anfängt, wo der vorhergegangene aufgehört hat. Der Lese- oder Schreibkopf muss dann nicht neu positioniert werden. Es kann kontinuierlich gelesen werden und kommt zu keiner Verzögerung.

Beim Schreiben eines Filmstreams auf einen nicht fragmentierten Speicher ist das der Fall. Bei vorhandener Fragmentierung kann dies vom Dateisystem erzwungen werden. Dafür unterstützen manche Dateisysteme die sogenannte Preallokation. Dabei wird ein beliebig großer sequentieller Bereich für Dateien reserviert. Das Dateisystem legt dann leere Dateien sequentiell an, die später beliebig gefüllt werden können. Die Größe der Dateien muss jedoch bei der Preallokation festgelegt werden.

Ein einmal sequentiell abgespeicherter Filmstream kann nun beliebig oft sequentiell wieder ausgelesen werden. Falls auf dem Datenspeicher in dieser Zeit keine weiteren Zugriffe erfolgen, entstehen auch keine zusätzlichen Kopfsprünge, es wird die bestmögliche Performance erreicht.

2.3 Latenz

Unter Latenz versteht man bei Festplattenzugriffen die Zeit, die es dauert bis ein angeforderter Block gelesen und an die Applikation zurückgegeben wurde. Durch viele Kopfsprünge erhöht sich die Latenz des Festplattenzugriffes. Da viele Applikationen in dieser Zeit blockierend warten, das heißt keine weiteren Operationen ausführen können, ist es wichtig die Latenz möglichst gering zu halten.

2.4 Cluster-Dateisysteme

Auf die physikalischen Blöcke einer Festplatte wird meist ein Dateisystem abgebildet, welches die Zuordnung zwischen physikalischen Blöcken der Festplatten und virtuellen Dateien im Betriebssystem herstellt. Bei einem SAN wird, um dabei Konflikte zwischen mehreren Clients im Netzwerk zu vermeiden, ein spezielles Clusterdateisystem wie Quantums StorNEXT oder CXFS von SGI verwendet.

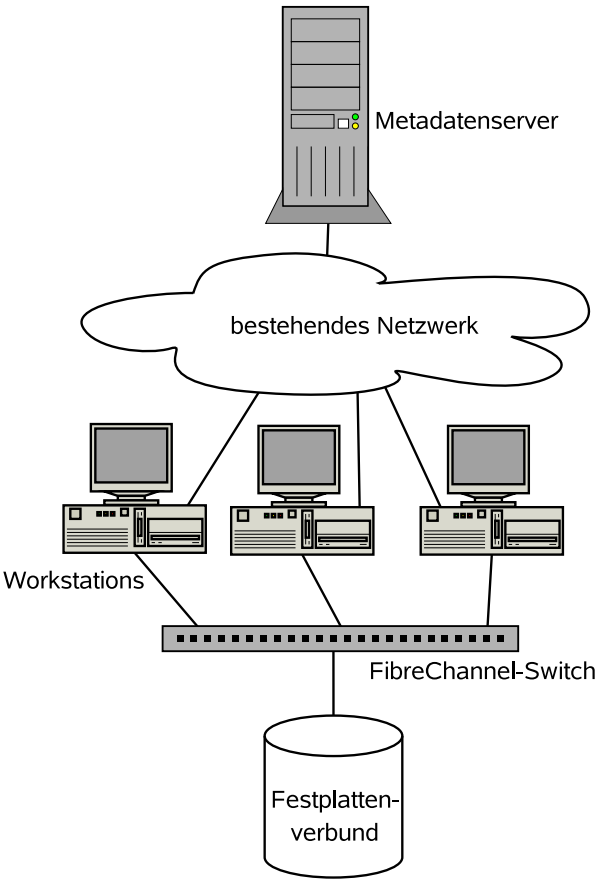

Abb. 4: *Beispiel eines Storage Area Networks mit 3 Workstations*

Diese Dateisysteme haben jeweils einen sogenannten Metadatenserver, an dem sich ein Client anmelden muss, bevor er auf den Datenspeicher zugreift. Dieser koordiniert den Zugriff und stellt sicher, dass keine Zugriffskonflikte oder Inkonsistenzen auftreten.

Bei StorNEXT liegen die Beschreibungsstrukturen nicht wie bei einem lokalen Dateisystem auf dem Datenträger, sondern werden ausschließlich auf dem Metadatenserver gehalten. Ein Zugriff ohne Kommunikation mit diesem ist deshalb nicht möglich.

CXFS ist eine Erweiterung für das Dateisystem XFS. Hier liegen die Beschreibungsstrukturen auf dem Datenträger. Ein einzelner Client kann deshalb auch ohne Metadatenserver auf das Dateisystem zugreifen. Sobald aber mehreren Clients zeitgleicher Zugriff ermöglicht werden soll, wird der Kontakt zu einem Metadatenserver notwendig.

2.5 I/O-Scheduling unter Linux

Moderne Betriebssysteme haben sogenannte I/O-Scheduler integriert. Diese sortieren die eingehenden I/O-Requests, so dass der Zugriff möglichst sequentiell erfolgt. Anzahl und Entfernung der Kopfsprünge werden minimiert, was die Performance erheblich steigern kann.

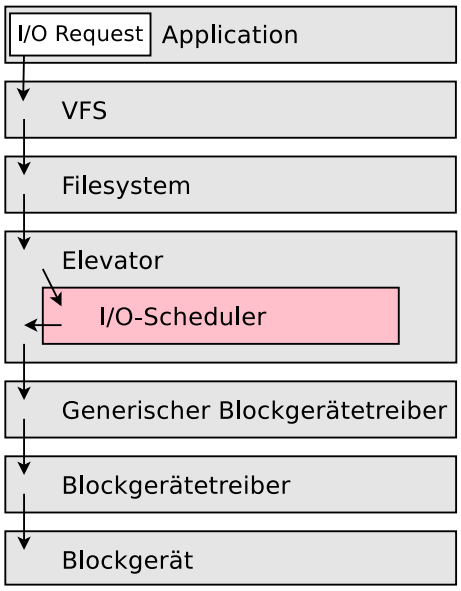

Abb. 5: *Der I/O-Stack unter Linux*

Der Kernel des Betriebssystems Linux hat bereits mehrere I/O-Scheduler integriert, jeweils für verschiedene Anwendungen und Typen von Datenträgern. Sie werden über die *Elevator*-Schnittstelle eingebunden und können während des Betriebes beliebig über das Sys-Filesystem ausgewechselt werden. Diese modulare Schnittstelle ermöglicht es, verschiedene Scheduler ohne großen Aufwand in den Betriebssystemkern zu integrieren und erleichtert das Entwickeln weiterer I/O-Scheduler (Siehe [Lov05], Seite 244 ff.). Die Elevator-Schnittstelle arbeitet zwischen dem Dateisystemtreiber und dem Treiber für das Blockgerät (Abb. 5). Deshalb ist sie weder vom Dateisystem noch von der Art des Blockgerätes abhängig und arbeitet für beide völlig transparent.

Alle im Linux-Kernel vorhandenen I/O-Scheduler versuchen eine möglichst sinnvolle und gleichmäßige Verteilung der Beantwortung von Anfragen der Applikationen zu erreichen. Diese Technik wird *Fair Scheduling* genannt (Siehe [Lov05], Seite 245 ff). Dies kann dazu führen, dass eine Applikation mit Echtzeitanforderung ihre benötigte Datenrate bei zu großer Auslastung des Datenspeichers nicht mehr erreicht.

2.6 Caches und Puffer

Der Cache ist ein schneller Speicher, in dem Daten zwischengespeichert werden können um Zugriffe auf die langsameren Festplatten zu puffern und dadurch zu beschleunigen.

Das Schreiben auf einen Datenspeicher kann über Caches gepuffert werden. Dabei nimmt der RAID-Controller die Requests des Clients an, schreibt sie zunächst in den Cache und gibt dem Client eine Bestätigung dass die Daten gespeichert wurden. Das tatsächliche Schreiben auf den Datenträger kann dann zu einem späteren Zeitpunkt erfolgen. Die Kapazität des Caches ist jedoch begrenzt, da dieser teurer ist als Festplattenspeicher. Sobald der Cache gefüllt ist, können Requests nur noch in der Geschwindigkeit verarbeitet werden, in der die Daten auf die Festplatten geschrieben werden.

In diesem Anwendungsfall sorgt der Cache lediglich dafür, einen geringen Jitter auszugleichen. Kurze Einbrüche der Datenraten können mit im Cache gehaltenen Daten kompensiert werden.

2.7 Überdimensionierung der Hardware

Um höhere Performance zu erlangen, ist es üblich sowohl die Anzahl der Speichergeräte als auch deren Anbindung zu erhöhen. Theoretisch gibt es hier keine Obergrenzen. Durch RAID-Verbünde können beliebig viele Speichergeräte zusammengefasst werden, da die aggregierte Datenrate der Summe der einzelnen Datenraten entspricht. Auch die Anbindung der Datenspeicher kann durch Multi-Pathing beliebig nach oben skaliert werden.

Dadurch können zum einen entstandene Performanceverluste ausgeglichen werden. Zum anderen kann aber auch Anwendungen eine gewisse Dienstgüte garantiert werden, da andere “störende” Anwendungen unmöglich in der Lage sind, die volle Kapazität auszunutzen.

Der Kostenaufwand ist hier besonders hoch. Besonders Techniken für die Anbindung der Datenspeicher wie InfiniBand oder FibreChannel sind mit enormen Kosten verbunden.

Auch hochwertigere Hardware kann Performanceprobleme oft lösen. Es existieren bereits hochwertige Speichergeräte, welche den Datenfluss intern optimieren und dadurch weit höhere Leistungen ermöglichen. Preislich liegen diese Geräte jedoch um ein vielfaches höher als Low-Cost-Speichergeräte, welche theoretisch die gleiche Leistung erbringen könnten. Das Garantieren von Bandbreite ist jedoch mit diesen Geräten ebenfalls nicht möglich.

2.8 DIOS-Framework

Der von Thomson entwickelte *Distributed I/O-Scheduler* (DIOS) ist ein auf dem Client-Server-Prinzip basierendes Framework, welches nach dem Prinzip des Zeitmultiplex-Verfahrens arbeitet. Einzelnen Rechnern wird für kurze Zeitabschnitte exklusiver Zugriff auf ein Speichergerät gegeben. Wenn ein Rechner sequentiell liest, kann er während seine Zeitabschnittes die maximal mögliche Datenrate erreichen.

Dazu müssen sich zunächst alle Clients am Server anmelden. Der Server gibt dann jeweils den Schedulern aller Rechner, außer dem der den Zugriff erhalten soll den Befehl alle Requests zurückzuhalten. Jeder Rechner bekommt nach dem Round-Robin-Prinzip kurz exklusiven Zugriff. Die Anzahl der Kopfsprünge wird dadurch minimiert und die Datenraten um bis zu 100% erhöht (Siehe [Fü07], Seite 61 ff).

Das Verhältnis der Bandbreitenverteilung pro Rechner kann über das Sys-Filesystem beeinflusst werden.

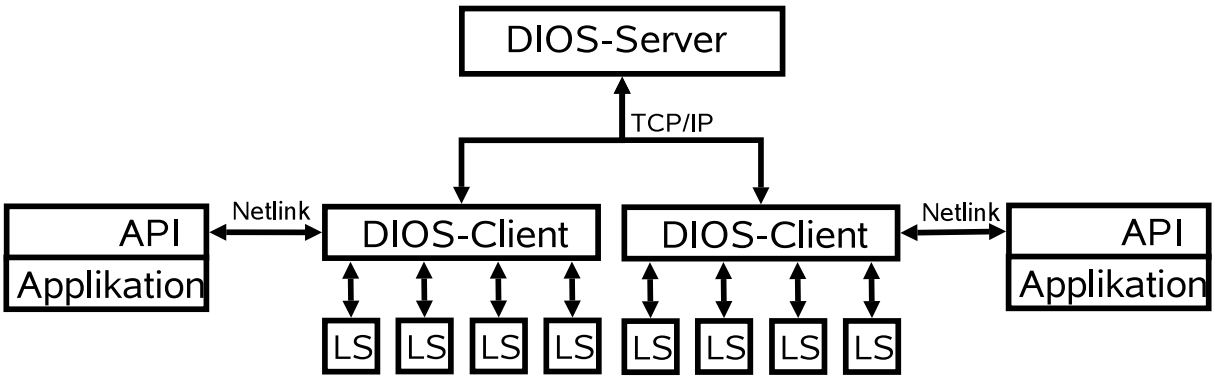

Abb. 6: *Schema des DIOS-Frameworks*

Clientseitig

Clientseitig gibt es zwei Kernel-Module: den *DIOS Local Scheduler* (LS) und den *DIOS-Client*. Der LS ist der eigentliche Scheduler und wird im Betriebssystem in den Elevator geladen. Ein SAN kann aus mehreren physikalischen Geräten bestehen, die erst auf Dateisystemebene zu einem logischen Laufwerk zusammengefasst werden. In diesem Fall muss für jedes Gerät eine Instanz des LS geladen werden.

Der Client ist sowohl für die Koordination der verschiedenen Instanzen des LS, sowie für die Netzwerkkommunikation zum Server und zu Applikationen zuständig.

Vom Server erhält er Befehle, ob er Requests durchlassen darf, oder sie blockieren muss um einem anderen Rechner exklusiven Zugriff zu erlauben.

Falls kein Server vorhanden ist, arbeitet der LS als NOOP-Scheduler, das heißt er reicht die Requests ohne weitere Scheduling-Maßnahmen weiter. Dieser Modus ist lediglich als Notfall-Lösung gedacht, um bei einer Störung den Zugriff auf das Blockgerät weiterhin zu ermöglichen.

Serverseitig

Der Server verteilt an alle angemeldeten Rechner nach dem Round-Robin-Prinzip Zeitscheiben. Während seiner Zeitscheibe hat der Rechner exklusiven Zugriff auf das Speichergerät. Er besteht aus 2 Kernel-Modulen, dem Server-Modul für die Verwaltung der Netzwerkverbindung und dem Globalen Scheduler für die Berechnung der Zeitscheiben.

3 Problematik und Anforderungen

In diesem Kapitel soll die in dem Anwendungsfall bestehende Problematik erläutert und die sich daraus ergebenden Anforderungen hergeleitet werden.

3.1 Aktueller Stand

In [Son06] ist gezeigt worden, dass mit Festplattenverbünden ohne interne Optimierung hohe Datenraten erreicht werden können. Dies trifft jedoch nur zu, wenn ein einzelner Datenstrom sequentiell gelesen oder geschrieben wird. Bei wahlfreien Zugriffen sinkt der Datendurchsatz aufgrund der Sprünge der Festplattenköpfe auf 50% und weniger.

In [Fü07] ist gezeigt worden, dass dieser Effekt durch ein Zeitmultiplexverfahren minimiert werden kann. Zu diesem Zweck wurde das DIOS-Framework entwickelt. Dies macht es möglich die Kopfsprünge zwischen mehreren sequentiellen Datenströmen auf ein Minimum zu beschränken und so wieder annähernd die volle Datenrate zu erreichen.

In einer Postproduktionsumgebung werden hauptsächlich sequentielle Datenströme gelesen und geschrieben. Aufgrund der weitaus niedrigeren Anschaffungskosten ist es wünschenswert ein nichtoptimiertes Gerät einem durch komplexe Logik für wahlfreie Zugriffe optimiertem Speichergerät vorzuziehen.

3.2 Problematik

Eine Applikation die sequentiell auf das SAN zugreifen will kann von anderen Applikationen behindert werden. Wenn mehrere Applikationen sequentiell zugreifen, stören sie sich gegenseitig. Durch Vermischung der Requests entspricht die erreichte Performance nicht mehr der von sequentiellen Datenströmen, sondern entspricht der Performance von wahlfreien Zugriffen.

Die Applikationen können dabei sowohl auf der selben als auch auf verschiedenen Workstations laufen. Aus diesem Grund ist der übliche Ansatz des I/O-Schedulings hier nicht ausreichend.

3.2.1 DIOS

Das DIOS-Framework optimiert die Zugriffe im SAN durch ein Zeitmultiplexverfahren. Dies geschieht jedoch lediglich zwischen den angemeldeten Rechnern. Auf einzelne Applikationen wird keine Rücksicht genommen. Es muss also vom Benutzer sichergestellt werden, dass auf jedem Rechner eine Applikation exklusiv auf das Speichergerät zugreift. Eine zweite Anwendung macht selbst wenn sie nur geringen Datendurchsatz erzeugt, den Effekt des DIOS-Frameworks zunichte. Der Betrieb von mehreren auf das Speichergerät zugreifenden Applikationen auf einer Workstation ist nicht vorgesehen.

Desweiteren wird davon ausgegangen, dass alle zugreifenden Rechner hohe Datendurchsätze erzielen. Wenn eine Workstation einen geringen Datendurchsatz erzeugt, wird für diese Applikation trotzdem ein Zeitschlitz mit exklusivem Zugriff reserviert, wodurch den anderen Applikationen Bandbreite verloren geht.

Der DIOS-Server ist in Form von zwei miteinander interagierenden Kernel-Modulen implementiert. Dadurch wird die Entwicklung und besonders die Fehlersuche erheblich erschwert. Außerdem führt es leichter zu Stabilitätsproblemen, da zum Beispiel ein Programmfehler in einem Kernel-Modul nicht nur die Applikation sondern das gesamte System abstürzen lassen kann.

Durch die Existenz von Netzwerkverbindungen im Kernel-Space könnte ein Angreifer von außen weit mehr Schaden anrichten als es bei einer Userspace-Applikation möglich wäre. Außerdem gibt es weniger Möglichkeiten die Applikation zusätzlich gegen solche Angriffe abzusichern als es im User-Space der Fall ist.

3.3 Anforderungen

Aus dieser Problematik ergeben sich nun mehrere Anforderungen.

3.3.1 Scheduling pro Applikation

Der Ansatz des Zeitmultiplexverfahrens im DIOS-Framework soll erweitert werden. Das Framework muss nicht nur zwischen Rechnern, sondern auch zwi-

schen auf der selben Workstation laufenden Applikationen unterscheiden. Dazu muss auf lokaler Seite eine weitere Ebene in der Schedulinghierarchie eingerichtet werden. Während der Server weiterhin für das Scheduling zwischen den Workstations zuständig ist, muss der DIOS-Client die auf den Workstations laufenden Applikationen verwalten und einzeln in den Scheduling-Mechanismus einbeziehen.

3.3.2 Quality of Service

Einer Anwendung mit Echtzeitanforderungen muss auf Anforderung eine gewisse Datenrate kontinuierlich garantiert zur Verfügung gestellt werden können, unabhängig von der übrigen Auslastung des Datenspeichers. Es muss also eine Möglichkeit gefunden werden, Applikationen angeforderte Mindestbandbreiten zuzuordnen und diese Applikationen dann bei Bedarf gegenüber anderen Applikationen zu bevorzugen. Zusätzlich sollen bestimmte Applikationen, die sequentiell lesen oder schreiben, aber keine Echtzeitanforderungen haben, nichtsequentiell zugreifenden Anwendungen gegenüber bevorzugt werden. Darduch wird das bei sequentiellen Zugriffen günstige Verhalten der Speichergeräte ausgenutzt.

3.3.3 Portierung des Servers in den Userspace

Um die Entwicklung zu vereinfachen und Flexibilität, Stabilität und Sicherheit zu erhöhen, soll der DIOS-Server in den Userspace portiert werden. Als positiver Nebeneffekt besteht dadurch die Möglichkeit, die Programmiersprache frei zu wählen und auf diverse öffentlich zugängliche Bibliotheken zurückzugreifen. Auch wird eine eventuelle zukünftige Erweiterung des Frameworks dadurch erleichtert.

4 Lösungsansatz

Dieses Kapitel erläutert wie diese Anforderungen durch eine Softwarelösung erfüllt werden können und wie diese umgesetzt werden kann.

4.1 Quality of Service

In dem zuvor beschriebenen Szenario tauchen im wesentlichen drei Arten von Applikationen auf. Zur einfachen Verwaltung werden diese jeweils in eine von drei Klassen eingeteilt.

- Prozesse, die Daten sequentiell lesen oder scheiben und dabei eine die eine bestimmte konstante Mindestbandbreite benötigen und diese über die API beantragt haben. Diese Prozesse werden im folgenden Realtime-Applikationen (RT) genannt. Es muss stets garantiert werden, dass RT-Prozesse Ihre geforderte Bandbreite erhalten. Lediglich kleinere Schwankungen in der Datenrate können durch Caches ausgeglichen werden (Siehe [Fü07], Seite 69).
- Der zweiten Klasse werden Prozesse zugeordnet, die ebenfalls sequentiell auf den Speicher zugreifen, jedoch keine Echtzeitanforderungen haben. Diese werden ähnlich wie RT-Prozesse behandelt, jedoch gibt es keine notwendige Datenrate die erreicht werden muss. Aus diesem Grund wird die nicht anderweitig verwendete Bandbreite gleichmäßig unter diesen Prozessen aufgeteilt. Diese Prozesse werden Non-Realtime-Applikationen (RT) genannt.
- Alle übrigen Prozesse, die keine Echtzeitanforderung haben und nicht sequentiell lesen, werden Best-Effort-Prozesse (BE) genannt. Dieser Klasse werden automatisch alle Prozesse zugeordnet, die sich nicht bei der API anmelden. Sie werden grundsätzlich gegenüber RT- und NRT-Prozessen benachteiligt. Falls gar keine oder nur RT-Prozesse angemeldet sind, wird sämtliche nicht von den RT-Prozessen beanspruchte Zeit für BE-Prozesse verwendet. Falls NRT-Prozesse aktiv sind erhalten die BE-Prozesse nur noch einen als Parameter übergebenen minimalen Zeitschlitz, standardmäßig 5% der Gesamtzeit. Der Rest wird an die NRT-Prozesse verteilt.

4.2 Zeitmultiplex-Verfahren

Da die Zugriffe der einzelnen Applikationen sequentiell erfolgen, ist ein Sortieren der Anfragen nicht erforderlich (Siehe [Fü07]). Kopfsprünge finden dann nur statt, wenn zwischen den Datenströmen zweier Applikationen gewechselt wird. Während eine Applikation exklusiven Zugriff auf den Speicher hat, ist die durch Kopfsprünge verlorene Bandbreite minimal. Wenn mehrere Applikationen zugreifen wollen, muss also jede Applikation jeweils für einen kurzen Zeitraum exklusiven Zugriff bekommen. Lediglich beim Wechseln zwischen zwei Applikationen findet dann ein Kopfsprung statt. Je häufiger gewechselt wird, desto mehr Bandbreite geht verloren.

Dies muss nun rechnerübergreifend zwischen allen Applikationen koordiniert werden um sicherzustellen, dass sich auch Applikationen auf verschiedenen Rechnern nicht gegenseitig behindern.

Da während des exklusiven Zugriffs einer Applikation die Zugriffe sämtlicher konkurrierenden Applikationen blockiert werden, muss sichergestellt werden, dass die blockierenden Pausen nicht zu lang werden, da dies zu einer hohen Latenz führen würde. Threads die blockierend auf ein Gerät zugreifen, "hängen" während sie auf eine Antwort des Gerätes warten, es können während dieser Wartezeit keine anderen Operationen ausgeführt werden. Wenn der Thread nicht-blockierend zugreift, besteht die Gefahr, dass er in einen Timeout läuft, deswegen davon ausgeht dass ein Zugriffsfehler aufgetreten ist und den Zugriff abbricht.

Daraus folgen zwei Gegebenheiten:

- Je kleiner die Zeitscheiben, desto geringer die durchschnittliche Performance.
- Je größer die Zeitscheiben, desto höher die Latenz.

Je nach Anforderung sind also größere oder kleinere Zeitscheiben sinnvoll, oder es muss ein gutes Mittelmaß gefunden werden. In welcher Größenordnung diese Werte sinnvoll sind, soll durch Messungen ermittelt werden.

4.3 Globale Koordination

Wie in [Fü07] gezeigt, genügt es, während eines sich ständig wiederholenden Zyklus, Zeitscheiben an alle Rechner zu verteilen während diese exklusiv zugreifen dürfen.

Die Länge der Zeitscheiben t_{slot} muss bei Anforderung von t_{needed} MB/s, bei einem Zyklus der Dauer t_{cycle} und einer physikalischen Anbindung von t_{total} MB/s immer

$$t_{slot} = bw_{needed} * \frac{t_{cycle}}{bw_{total}}$$

sein. So können während dieser Zeitscheibe soviel Daten gelesen werden, wie pro Zyklus gelesen werden müssen, so dass die Zugriffe der Applikation für den Rest des Zyklus blockiert werden können. Die durchschnittliche Datenrate entspricht dann der geforderten.

Da die Applikationen auf verschiedenen Rechnern laufen, ist es notwendig auch die Scheduler-Module rechnerübergreifend zu koordinieren. Wie bereits im DIOS-Framework gezeigt, eignet sich dafür eine Client-Server-Architektur. Das heißt, der Scheduler auf jedem Rechner besitzt eine Client-Komponente, die sich über ein TCP/IP-Netzwerk zu einem zentralen Server verbindet, und diesem Informationen über die lokalen Applikationen überträgt.

Daraufhin berechnet der Server die Scheduling-Intervalle für sämtliche Clients und sendet sie an die Clients, welche die Einhaltung sicherstellen.

Dabei ist es absolut notwendig, das alle Clients des SAN an diesem Server angemeldet sind. Ein einzelner nicht angemeldeter Rechner der ohne Einschränkungen auf das SAN zugreifen kann, kann den gesamten Datenfluss kontinuierlich stören. Der exklusive Zugriff einzelner Applikationen kann dann nicht mehr sichergestellt werden.

4.4 Kommunikation

Der Server muss also den Client-Rechnern je nach Bedarf Zeitscheiben zuweisen. Der Client unterteilt diese dann ggf. erneut, damit jede Applikation eine exklusive Zeitscheibe erhält. Der lokale Scheduler entscheidet dann letztendlich

ob eine Applikation sofort auf den Datenspeicher zugreifen darf oder zunächst blockiert wird.

Um ungewollten konkurrierenden Zugriff mehrerer Applikationen zu vermeiden, erfolgt die Synchronisation mittels eines Tokens. Der Server gibt das Token zusammen mit der Zeit der Berechtigung in Millisekunden nacheinander jeweils einem Client, dieser gibt es dem Server nach Ablauf seiner Zeitscheibe zurück.

Der Client wiederum gibt das Token an den lokalen Scheduler weiter. Dazu teilt er ihm mit, welcher Prozess zugreifen darf, und wie lange seine Zeitscheibe ist. Sofern eine Applikation eine feste Bandbreite gemakelt hat, wird dem Token ebenfalls die Datenmenge, die die Applikation in der Zeit lesen soll mitgegeben. Falls die Datenmenge vor Ablauf der Zeitscheibe gelesen worden ist, kann das Token frühzeitig zurückgegeben werden, wodurch die übrige Zeit anderen Prozessen zur Verfügung gestellt werden kann.

5 Implementierung

In diesem Kapitel wird auf die technischen Details der Implementierung eingegangen und es werden entstandene Probleme und Besonderheiten erläutert.

Die Implementierung erfolgt teilweise auf Basis des DIOS-Frameworks von Thomson. Jedoch müssen hier einige Teile ersetzt oder erweitert werden.

5.1 Lokaler Scheduler

Der lokale Scheduler hat die Aufgabe, zu entscheiden wann Requests einer Applikation an den Blockgerätetreiber weitergegeben werden. Die eingehenden Requests werden zunächst in FIFO-Queues einsortiert. Wenn eine Applikation sich als RT- oder NRT-Applikation anmeldet, wird für sie eine dedizierte Queue angelegt. Außerdem gibt es eine gemeinsame Queue für alle BE-Prozesse. Jeder Prozess der sich nicht als RT- oder NRT-Prozess angemeldet hat, wird zunächst als BE-Prozess betrachtet. Ein Prozess kann seine Priorität jederzeit beliebig zwischen RT, NRT und BE wechseln.

Der lokale Scheduler unterscheidet die Queues nicht zwischen RT- und NRT-Prozessen, da beide exklusiven Zugriff erhalten. Lediglich BE-Prozesse müssen unterschieden werden, da deren Zugriffe nicht sortiert oder priorisiert werden müssen, also gemischt erfolgen.

Vom Dios-Client erhält der Scheduler nun in regelmäßigen Abständen ein Token und

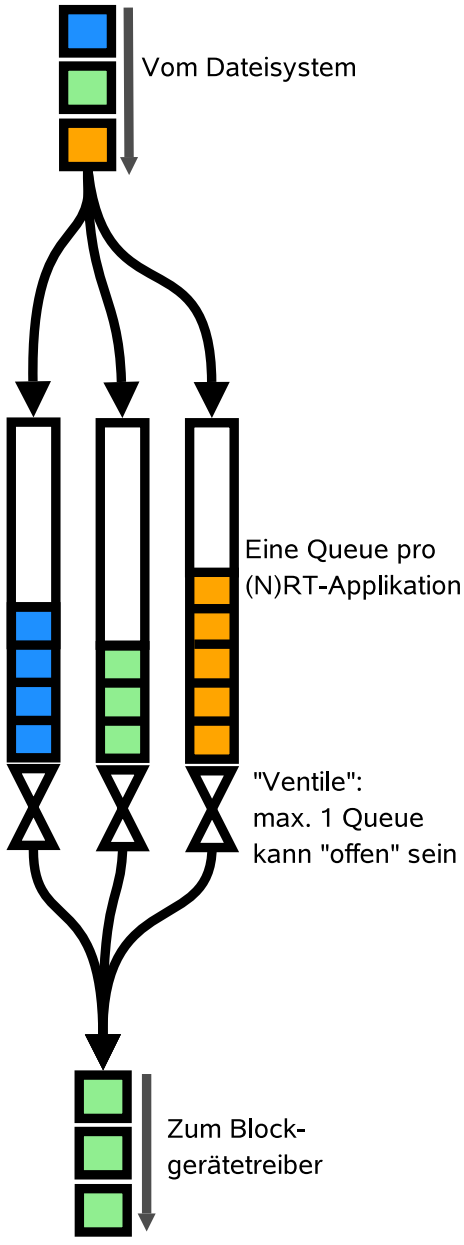

Abb. 7: *Mehrere Queues im lokalen Scheduler*

zusätzlich die Information für welchen Prozess dieses Token bestimmt ist und wie lange dieses gültig ist. Die Identifikation erfolgt anhand der Prozess-ID (PID).

Solange der Scheduler dieses Token hält, werden die Requests aus der entsprechenden Queue an den Blockgerätetreiber weitergereicht. Zusätzlich zu den prozessgebundenen Tokens, gibt es BE-Tokens. Hält der lokale Scheduler ein solches, wird die BE-Queue abgearbeitet. Hier wird keine Prozess-ID mit übergeben, aber ebenfalls die Dauer der Gültigkeit des Tokens. Nach Ablauf der Gültigkeit wird das Token an den Client zurückgegeben. Dann sind die Zugriffe zunächst wieder blockiert. Hat ein Prozess eine bestimme Bandbreite beantragt, so wird mit dem Token zusätzlich die Information übergeben, wieviel Byte der Prozess während der Zeitscheibe lesen muss, um auf seine Datenrate zu kommen. Falls er diese Datenmenge vor Ablauf der Zeitscheibe erreicht, wird das Token frühzeitig zurückgegeben, und die übrige Zeit an andere Applikationen verteilt.

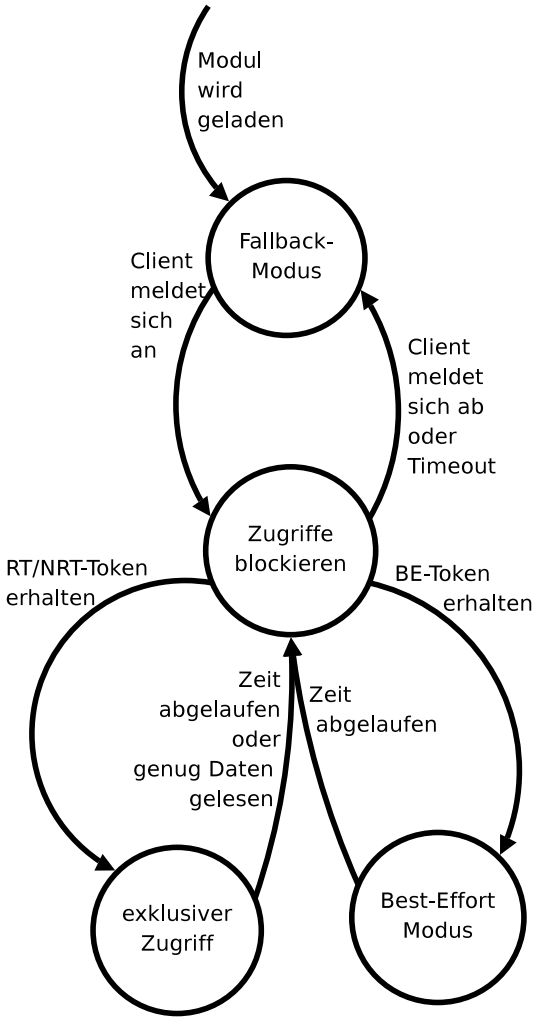

Abb. 8: *Zustandsdiagramm des lokalen Schedulers*

5.1.1 Fallback-Modus im Fehlerfall

Der lokale Scheduler verfügt außerdem über einen Fallback-Modus, falls ein unerwarteter Fehler auftritt. In diesen wechselt er, wenn das Client-Modul nicht geladen ist, oder wenn bei Kommunikation mit diesem ein Fehler auftritt.

Dies soll verhindern, dass im Fehlerfall weiterhin Zugriff auf den Datenspeicher möglich ist, und nicht wie im Normalbetrieb alle Zugriffe blockiert werden, wenn kein Token vorhanden ist.

In diesem Modus arbeitet der Scheduler so wie der NOOP-Scheduler des Linux-Kernels. Es werden also alle Requests in die BE-Queue einsortiert und kontinuierlich abgearbeitet. Sämtliche Vorteile des DIOS-Frameworks gehen hierbei jedoch natürlich verloren.

5.1.2 Kontextloser Betrieb

Eine größere Schwierigkeit ist, dass der lokale Scheduler keinen eigenen Kontext besitzt, das heißt er besteht lediglich aus Funktionen und Daten die im Kernel-Space abgelegt sind. Er hat keine eigenen Prozesse oder Threads wie man es von gewöhnlichen Applikationen kennt. Deshalb kann der Scheduler auch keine Funktionen selbstständig ausführen, sondern muss von anderen Threads, die einen Kontext haben angestoßen werden. Der Elevator ruft die ihm bekannten Funktionen nach Bedarf auf. Zwei dieser Funktionen sind dabei für das Verständnis von großer Wichtigkeit:

- *dios_add_request* wird aufgerufen, wenn eine Applikation einen neuen Request an das Blockgerät stellt
- *dios_dispatch* wird aufgerufen, wenn der Blockgerätetreiber einen neuen Request anfordert (Siehe [QK05]). Wird ihm daraufhin ein Request zurückgegeben, fordert er einen weiteren an. Wird ihm nichts zurückgegeben, geht er davon aus dass keine Requests vorhanden sind und stellt keine weiteren Anfragen, bis wieder ein *dios_add_request* ausgeführt wurde.

Das hat den negativen Effekt, dass wenn der lokale Scheduler die Requests zurückhält, um Applikationen auf anderen Rechnern Vorrang zu geben, der Blockgerätetreiber davon ausgeht dass keine weiteren wartenden Requests vorhanden sind und sich deshalb bis zum nächsten Aufruf von *dios_add_request* schlafen legt.

Bekommt der lokale Scheduler nun ein Token, kann er zwar die wartenden Requests wieder durchlassen, allerdings fordert der Blockgerätetreiber weiterhin keine Requests an.

Aus diesem Grund wird in jeder Instanz des lokalen Schedulers sobald der Zugriff blockiert wird ein Timer angelegt, dem ein Zeiger auf die Datenstruktur des Schedulers übergeben wird. Dieser besitzt einen eigenen Kontext innerhalb des Kernels und er überprüft periodisch in jeder Millisekunde ob ein neues Token vorhanden ist und führt gegebenenfalls *dios_dispatch* mit dem Zeiger auf die Datenstruktur des Schedulers als Parameter aus. Während im normalen Betrieb passiv auf Anfragen des Treibers gewartet wird, werden ihm die Requests nun aktiv ohne Aufforderung zugeschoben.

5.2 Client

Der DIOS Client meldet sich beim Laden am Server an. Sobald eine Applikation über die API Bandbreite makelt oder Priorität beantragt, leitet der Client die Anfrage an den Server weiter und speichert bei positiver Antwort die Prozess-ID (pid) des Clients sowie ggf. die reservierte Bandbreite in einer Liste ab. Erhält der Client nun ein Token vom Server, was dem jeweiligen Rechner für einen bestimmten Zeitraum den Zugriff auf das Storage erlaubt, teilt der diesen Zeitraum auf die bekannten Applikationen auf und gibt das Token mit diesen Informationen an den Scheduler weiter.

Im Gegensatz zum lokalen Scheduler, startet der Client einen eigenen Thread. Er läuft also in einem eigenen Kontext im Kernelspace. Dies ist für die Netzwerkkommunikation notwendig, nur so kann er ständig sofort auf eintreffende Nachrichten reagieren.

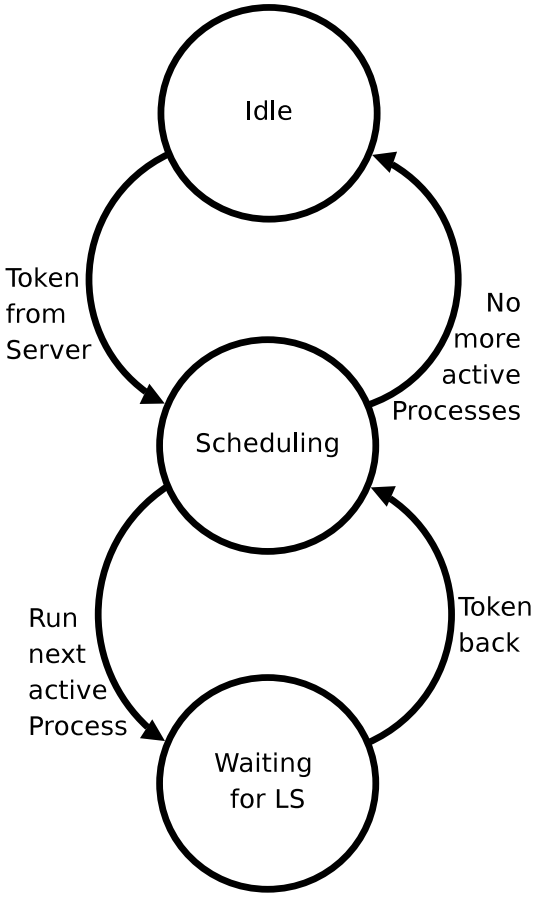

Abb. 9: *Zustandsdiagramm des Clients*

Die Kommunikation über die DIOS-API läuft asynchron in einem separaten Kernel-Thread. Dadurch ist diese von den Zuständen des Clients abgekoppelt und kann jederzeit parallel erfolgen.

5.2.1 parallele Datenhaltung

Da der lokale Scheduler und der Client keinen gemeinsamen Speicherbereich haben, müssen die Informationen über priorisierte Applikationen in beiden Modulen gespeichert werden.

Die Applikationen melden sich nur beim Client an, dieser muss es also dem lokalen Scheduler mitteilen, wenn neue Applikationen hinzukommen oder vorhandene sich abmelden.

Wenn eine Applikation sich beendet ohne sich vorher abzumelden, oder sogar abstürzt würden Client und Scheduler dies nicht merken, das heißt, die Datensätze würden für immer im Speicher vorhanden sein. Deshalb startet der Client einen Kernel-Timer, welcher alle zehn Sekunden überprüft ob alle Prozesse noch vorhanden sind, und im Falle das einer fehlt, dessen Daten löscht und gegebenenfalls dem Server mitteilt, dass die beantragte Datenrate wieder freigegeben werden soll.

5.3 Globaler Scheduler

Der *DIOS Global Scheduler* besteht aus einer in C++ implementierten Server-Anwendung.

Die Grundeinstellungen können als Parameter beim Starten übergeben werden. Die wichtigste Einstellung ist dabei die physikalische Bandbreite des Speichernetzes. Optionale Parameter sind die Dauer eines Zyklus und die Portnummer für die Netzwerkkommunikation.

Der Server arbeitet mit mehreren Threads. Für jeden verbundenen Client gibt es einen Thread, und außerdem einen Master-Thread, welcher für das eigentliche Scheduling zuständig ist. Wenn ein Client $bw_{requested} MB/s$ an Bandbreite anfordert, wird berechnet ob noch genügend Bandbreite frei ist. Die Bedingung dafür ist

$$bw_{total} - bw_{used} >= bw_{requested}$$

wobei bw_{total} der gesamten physikalischen Bandbreite entspricht und bw_{used} der bereits an andere Prozesse vergebenen. Die Anfrage wird dann entsprechend mit einem ACK bestätigt oder mit einem NACK abgelehnt. Falls das makeln erfolgreich war, wird die beantragte Datenrate in einer Liste abgelegt. Wenn ein Client NRT-Applikationen hat, teilt er dem Server lediglich die Anzahl selbiger mit. Diese wird ebenfalls in der Client-Liste gespeichert.

5.3.1 Scheduling

Zunächst bekommt nun jeder Rechner mit RT-Applikationen einen Zeitschlitz von

$$t_{slot} = \frac{bw_{needed}}{min(bw_{client}, bw_{storage})} * t_{cycle} * 1,05$$

Das entspricht der benötigten Zeit um die Daten zu lesen, die während des gesamten Zeitraumes t_{cycle} gelesen werden müssen, plus 5% um einen eventuell durch die Kommunikation entstehenden Zeitverlust auszugleichen (Siehe [Fü07], Seite 54).

Die restliche Zeit minus 5% wird nun durch die Anzahl der NRT-Applikationen geteilt und jeder Client mit NRT-Applikationen erhält eine der Anzahl selbiger entsprechender Zeitscheibe. Die restliche Zeit wird für BE-Applikationen verwendet. In diesem Zeitschlitz können die BE-Applikationen aller beteiligten Rechner gleichzeitig auf den Datenspeicher zugreifen. Der Zeitschlitz für BE-Applikationen ist also $t_{cycle} * 0,05$ wenn NRT-Applikationen vorhanden sind. Falls nur RT-Applikationen vorhanden sind, beträgt er $t_{cycle} - t_{rt}$.

Wenn alle BE-Tokens von den Clients zurückgesendet worden sind beginnt der Zyklus von vorne.

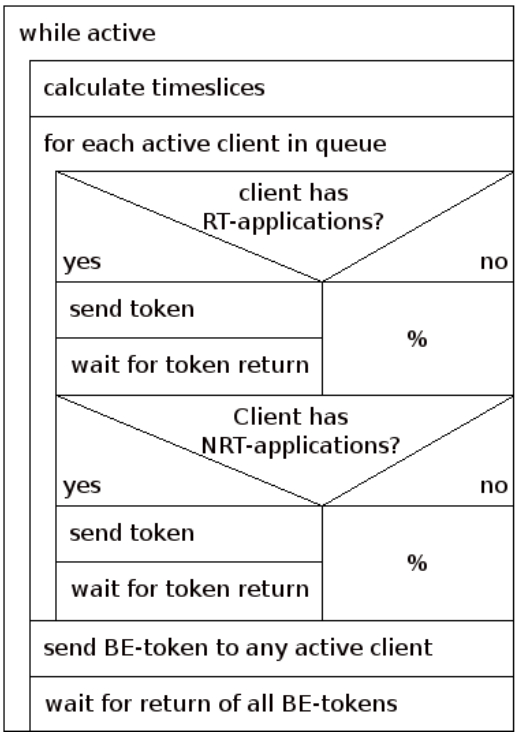

Abb. 10: *Struktogramm des Servers*

5.3.2 Überwachen der Aktivität

Um einem Client der Priorität beantragt hat, aber nicht auf den Datenspeicher zugreift nicht überflüssigerweise Zeitscheiben zuzuweisen, existiert der sogenannte *Idle-Mechanismus*. Sobald eine Applikation einen neuen Request absetzt, jedoch kein Token hält - der Request also zunächst zurückgehalten wird, wird dies dem DIOS-Server mitgeteilt. Dies geschieht jedoch höchstens einmal pro Zyklus, um überflüssigen Overhead zu vermeiden.

Wenn ein Client innerhalb von 3 aufeinanderfolgenden Zyklen keinen Request absetzt, wird er vom DIOS-Server in den *Idle-Modus* gesetzt. Dabei bleibt das Anrecht auf eine bereits beantragte Priorität zwar erhalten, doch bei der Vergabe der Tokens wird dieser Client nicht mehr berücksichtigt und es steht mehr Bandbreite für andere Clients zur Verfügung.

Sobald ein Client einen neuen Request absetzt, wird er wieder in die Liste der aktiven Clients aufgenommen.

5.4 Kommunikation

5.4.1 Client und Server

Für die Kommunikation zwischen Client und Server wird ein einfaches, auf TCP/IP basierendes Protokoll verwendet. Ein Paket besteht dabei jeweils aus den beiden Werten *command* und *value*. Dies ist für die benötigte Kommunikation ausreichend. Tabelle 2 listet diese Befehle im Detail auf.

Befehl	Parameter	Erläuterung
DIOS_REG	phy_bw	Anmeldung am Server, Parameter: Physikalische Bandbreite des Clients
DIOS_APPLY	bw	Beantragen von *bw* MB/s
DIOS_ACK	bw	Bestätigung von *bw* MB/s
DIOS_NACK	avail_bw	Ablehnung, weil nur noch *avail_bw* MB/s verfügbar sind
DIOS_NRT_COUNT	clientcnt	Client informiert Server über die Anzahl der zu priorisierenden Clients
DIOS_REQ		Eine Applikation des Clients will auf das Blockgerät zugreifen
DIOS_RT_TOKEN	timeslice	RT-Applikationen dürfen für *timeslice* ms zugreifen
DIOS_RT_AMOUNT	amount	RT-Applikationen sollen maximal *amount* MB lesen
DIOS_NRT_TOKEN	timeslice	NRT-Applikationen dürfen für *timeslice* ms zugreifen
DIOS_BE_TOKEN	timeslice	BE-Applikationen dürfen für *timeslice* ms zugreifen
DIOS_TOKEN_BACK		Rückgabe des Tokens
DIOS_MODE	mode	*nicht mehr benutzt*
DIOS_ERROR	errorcode	*nicht implementiert*
DIOS_UNREG		*Abmelden vom Server*
DIOS_HEARTBEAT	random	*nicht implementiert*

***Tab. 2:** Übersicht der Befehle des Netzwerkprotokolls*

Sämtliche Befehle und Parameter lassen sich als *unsigned long integer* codieren,

wodurch ein Paket nur 16 Bytes an Daten enthält. So wird der durch die Kommunikation entstehende Overhead möglichst gering gehalten.

5.4.2 Client und Applikation

Applikationen können über ein sogenannte Application Programming Interface Priorität (API) oder Bandbreite beim Client anfordern. Der Client leitet die Anfrage an den Server weiter, und gibt die Rückmeldung des Servers wiederum an die Applikation zurück.

Die API stellt 4 Funktionen zur Verfügung:

dios_apply_bw(bw);

Beantragen von *bw* MB/s

dios_apply_bw_external(bw, pid);

Beantragen von *bw* MB/s für den Prozess mit der Prozess-ID *pid*.

dios_release_bw();

Freigeben der gemakelten Bandbreite

dios_release_bw_external(pid);

Freigeben der gemakelten Bandbreite des Prozesses mit der Prozess-ID *pid*.

Um nur hohe Priorität ohne garantierte Bandbreite zu beantragen wird als Bandbreitenparameter -1 übergeben.

Die external-Funktionen sind gedacht um Priorität für proprietäre Applikationen, denen Quellcode nicht zur Verfügung steht zu beantragen.

Die Kommunikation zwischen der API und dem Client erfolgt über einen Netlink-Socket.

5.4.3 Client und lokaler Scheduler

Der lokale Scheduler macht seine Funktionen für andere Module benutzbar indem er sie exportiert (Siehe [QK06], Seite 264). Diese können dann vom Client aufgerufen werden.

Deswegen kann der Client nur geladen werden, wenn der lokale Scheduler bereits läuft. Jeder dieser Funktion setzt ein für alle Instanzen des Schedulers sichtbares Flag, worauf dann jede einzelne Instanz reagieren kann.

Der Client übergibt den Instanzen des lokalen Schedulers nach dem Laden eine Referenz auf eine Callback-Funktion. Diese kann kann von den Instanzen aufgerufen werden um mit dem Client zu kommunizieren. Da die Rückgabe des Tokens die einzige benötigte Form der Kommunikation vom Scheduler zum Client ist, ist ein Netlink-Socket nicht noetig.

6 Tests

In diesem Kapitel wird beschrieben, auf welche Weise das Framework getestet wurde, welche Ergebnisse dabei erzielt wurden und wie diese zu deuten sind.

6.1 Versuchsaufbau

Zum Testen stehen drei Speichereinheiten vom Typ *HUGE MediaVault 4210* der Firma *Ciprico* zur Verfügung. Diese haben jeweils zwei RAID-Controller die jeweils fünf Festplatten verwalten. Jeder RAID-Controller ist über eine FibreChannel-Leitung an einen FibreChannel-Switch angebunden. An diesen Switch sind vier PC-Workstations über je drei FibreChannel-Leitungen angeschlossen (Abb. 11). Eine FibreChannel-Leitung erreicht einen Datendurchsatz von bis zu 400 MB/s, das ergibt einen maximalen Durchsatz von 1,2 GB/s pro Workstation und 2,4 GB/s für die Festplatten. Der maximale Durchsatz eines *HUGE MediaVault* wird vom Hersteller mit 500 MB/s beziffert, aggregiert können also lediglich ca. 1,5 GB/s erreicht werden. Da das DIOS-Framework jedoch stets einem einzelnen Client exklusiven

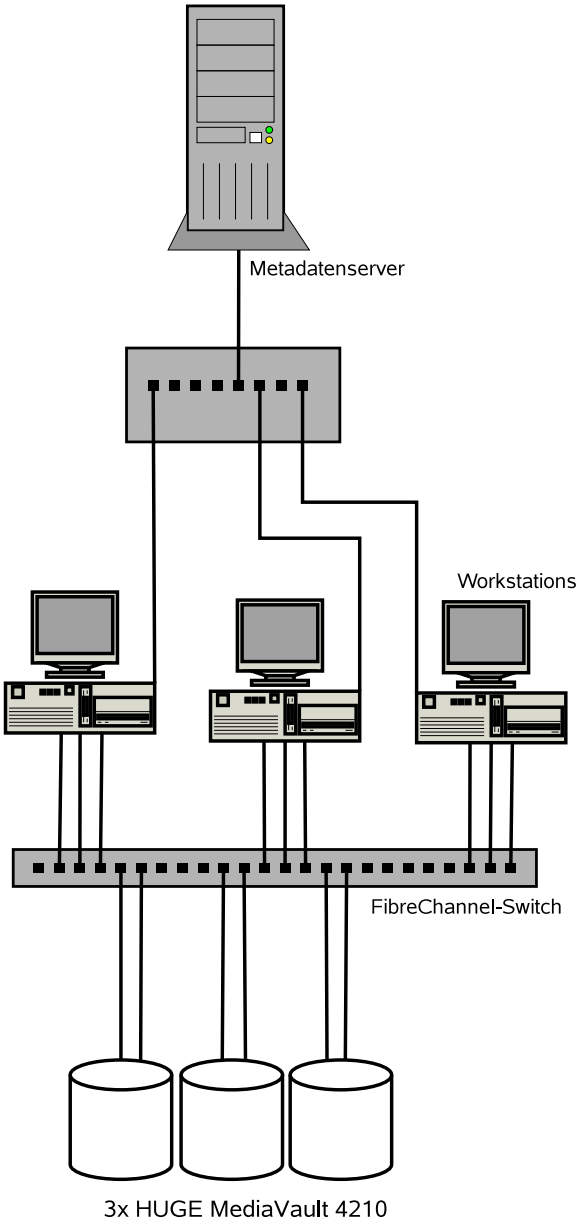

Abb. 11: *Skizze des Versuchsaufbaus*

Zugriff erlaubt, kann die Bandbreite der Clients nicht aggregiert werden. Der maximal zu erreichende Datendurchstatz entspricht deswegen dem schwächsten Glied in der Kette, also 1,5 GB/s.

Als Dateisystem wird das *StorNEXT FS* der Firma Quantum verwendet.

Dies stellt einen Teil einer typischen Arbeitsumgebung für die Postproduktion in der digitalen Filmproduktion dar.

6.2 Testverfahren

Zum Testen wird hauptsächlich das von Thomson entwickelte Benchmarking-Tool *Sanbs* verwendet. Dieses ist speziell auf Performance-Messungen von Storage Area Networks ausgerichtet und unterstützt sequentielles Lesen und Schreiben von beliebig großen Datenmengen mit sowohl höchstmöglicher Geschwindigkeit, als auch mit einer festgelegten konstanten Datenrate, um zum Beispiel einen Filmstream simulieren zu können. Dazu gibt es den sogenannten Sanbs-Master der mehrere Instanzen über eine Netzwerkverbindung synchronisiert. Die Sanbs-Instanzen melden sich beim Sanbs-Master an und bekommen dann zeitgleich ein Signal zum Start der Messung. Dieses Verfahren ermöglicht es sicherzustellen, dass während der gesamten Messung alle Instanzen parallel zugreifen und keine Überlappungen entstehen. Dies garantiert eine höchstmögliche Messgenauigkeit.

Es spielt dabei keine Rolle ob die Sanbs-Instanzen auf einem oder verschiedenen Rechnern gestartet werden. Es gibt also keine Beschränkungen an möglichen Test-Kombinationen. Des weiteren unterstützt Sanbs über die API des Frameworks Priorität oder Echtzeit-Bandbreite zu beantragen.

Bei den Tests soll folgendes herausgefunden werden:

- Wie lang dürfen die Zyklen sein, um ein gutes Mittelmaß zwischen Verringerung der Datenrate und Latenz zu bekommen?
- Wird die maximal mögliche Datenrate annähernd erreicht, auch wenn mehrere Applikationen von verschiedenen oder gleichen Workstations auf den Datenspeicher zugreifen?

- Bekommen priorisierte Applikationen ihre beantragten Datenraten auch, wenn andere Applikationen versuchen sehr schnell zu lesen?

6.3 Referenzmessung

Um die maximal mögliche physikalische Datenrate zu bestimmen wird zunächst ein einzelner sequentieller Datenstrom mit dem NOOP-Scheduler gelesen. Dabei ergibt sich eine konstante Datenrate von etwa 1150 MB/s. Der Verlust von ca. 50 MB/s entsteht vermutlich durch Overhead des Netzwerkprotokolls und lässt sich nicht verhindern.

6.4 Ergebnisse

Bei allen Messungen werden ausschließlich sequentielle Datenströme gelesen.

6.4.1 Variation der Zyklusdauer

Ein wichtiger Parameter des DIOS-Servers ist die Zyklusdauer. Ein Zyklus wird immer unter den Clients aufgeteilt. Beim Wechsel zwischen zwei Clients findet zwangsläufig ein Kopfsprung statt. Bei vier Clients und einem Zyklus von einer Sekunde entstehen also vier Kopfsprünge pro Sekunde. Weitere Kopfsprünge entstehen während des BE-Zeitschlitzes. Dieser ist jedoch sofern mindestens ein NRT-Client vorhanden ist immer 5%, also nicht von der Zyklusdauer abhängig[1].

Eine kurze Zyklusdauer führt also zu geringerer Performance, da mehr Kopfsprünge auftreten. Eine lange Zyklusdauer führt zu hoher Latenz, da die Pausen zwischen den Zugriffen einer Applikation länger sind.

Die Zyklusdauer kann der Anwender dem DIOS-Server als Parameter übergeben. Standardmäßig beträgt sie eine Sekunde. Um eine optimale Dauer zu finden, wird nun in einer Messreihe über eine Zyklusdauer von 100 Millisekunden bis 5 Sekunden iteriert.

Getestet wird mit zwei Sanbs-Instanzen auf zwei verschiedenen Workstations. Die Ergebnisse in Bild 12 zeigen, dass die Datenraten bis zu einer Zyklusdauer von 500 ms konstant sind. Bei kürzeren Zyklen bricht sie jedoch rapide ein. Zum Vergleich sind die besten unter vergleichbaren Bedingungen gemessenen Werte der Linux-Scheduler *noop* (425 MB/s) und *anticipatory* (506 MB/s) eingezeichnet. Selbst bei einer Zyklusdauer von 200 ms liegen die Ergebnisse noch über den des Anticipatory-Schedulers. Für Anwendungen bei denen geringe Latenz wichtiger ist als hoher Datendurchsatz können kürzere Zyklusdauern verwendet werden.

Für alle folgenden Messungen wurde eine Zyklusdauer von 1000 ms gewählt.

[1] Vgl. Kapitel 5.3.1

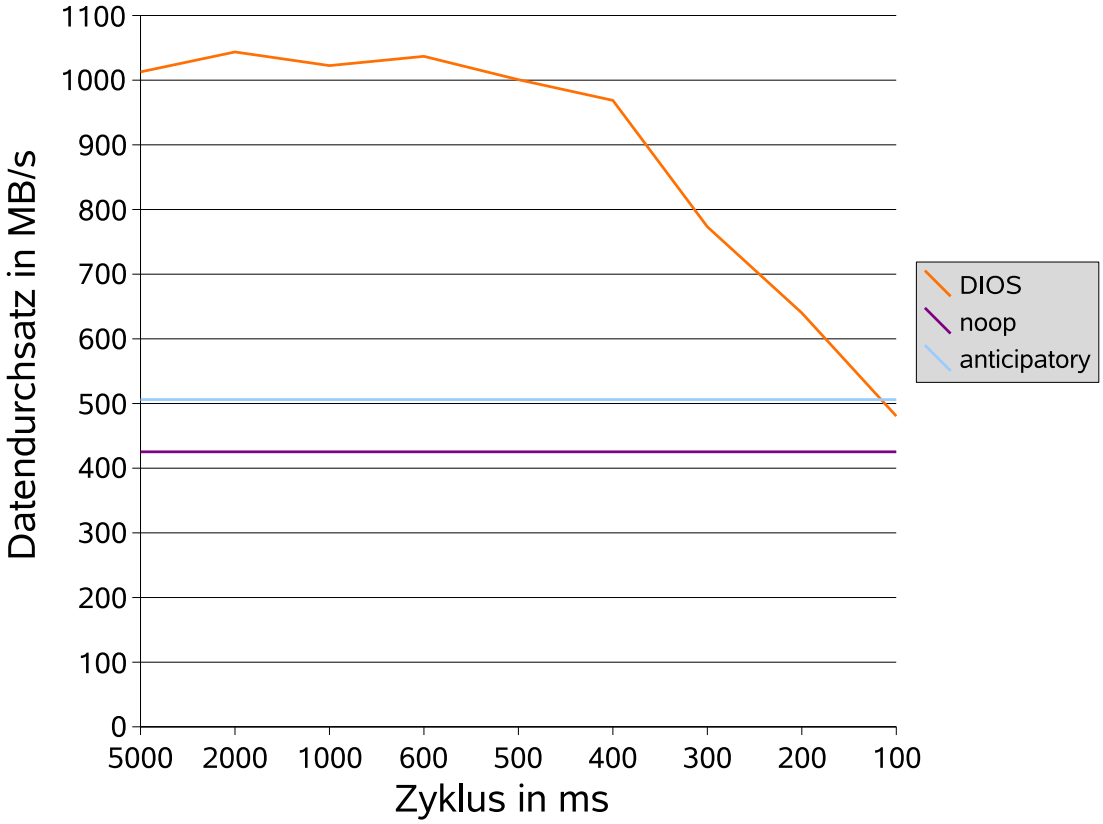

Abb. 12: *Tests mit verschiedenen Zykluslängen*

6.4.2 Sicherstellen der ursprünglichen Funktion

Nun soll überprüft werden, ob die ursprüngliche Funktion des DIOS-Frameworks weiterhin erfüllt wird, also ob durch den zusätzlichen Rechenaufwand eine Verschlechterung erfolgt ist.

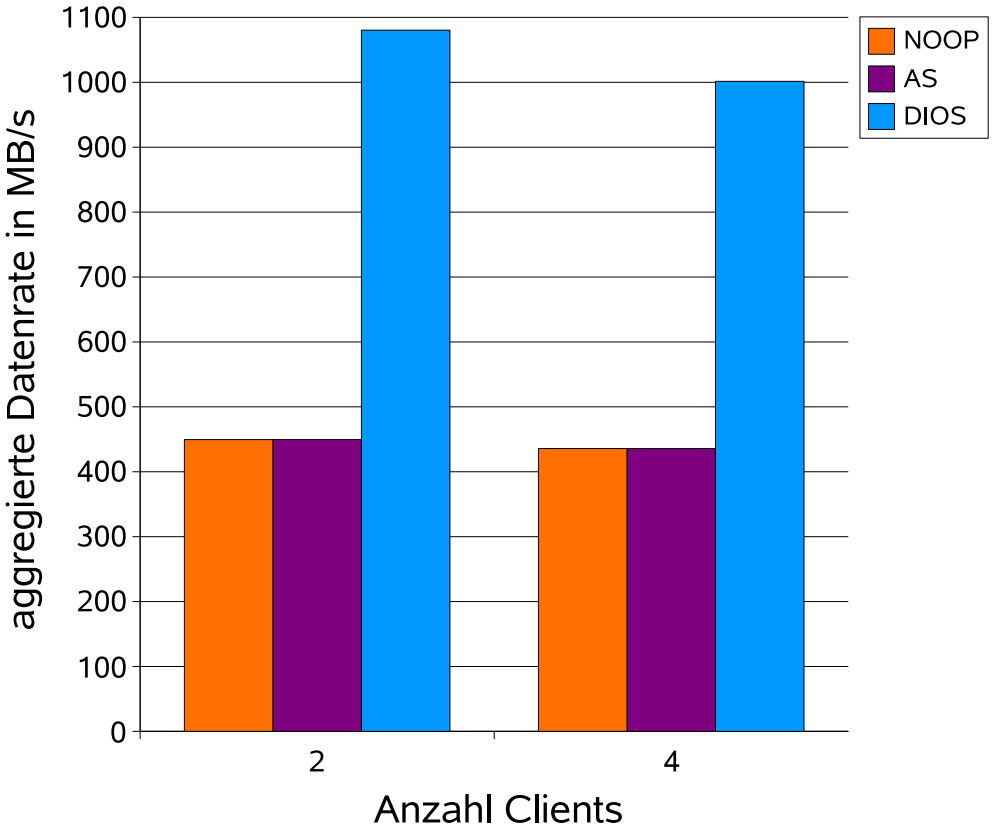

Abb. 13: *Vergleich zwischen DIOS und den Linux-Schedulern* noop *und* anticipatory.

Dazu werden insgesamt zwei Messreihen durchgeführt, eine mit zwei und eine mit vier Applikationen. Eine Applikation läuft dabei immer auf einem dedizierten Rechner. In jeder Messungen sollen die Applikationen parallel für 5 Minuten sequentiell Daten vom SAN lesen, so dass die Zugriffe interferieren. Das ganze wird mit 3 verschiedenen I/O-Schedulern getestet: Der NOOP-

Scheduler, welcher alle Requests ohne Scheduling weiterleitet, der effiziente Anticipatory-Scheduler und das DIOS-Framework. Dabei werden die Applikationen im NRT-Modus gestartet um höchstmögliche Datenraten und eine möglichst gleichmäßige Verteilung zwischen den Applikationen zu erreichen.

Wie in Abbildung 13 zu sehen ist hat ein Scheduler auf lokaler Seite keinen messbaren Effekt. Mit dem DIOS-Framework wird jedoch mehr als die doppelte Datenrate erreicht. Die entspricht den Ergebnissen aus [Fü07]. Diese Messung zeigt lediglich, dass keine Verschlechterung gegenüber dem vorherigen Stand erfolgt ist.

6.4.3 Applikationsbezogenes Scheduling

Nun soll gezeigt werden, dass die gleichen Ergebnisse auch erzielt werden, wenn mehrere Applikationen auf einer Workstation aktiv sind und konkurrierend Datenströme lesen.

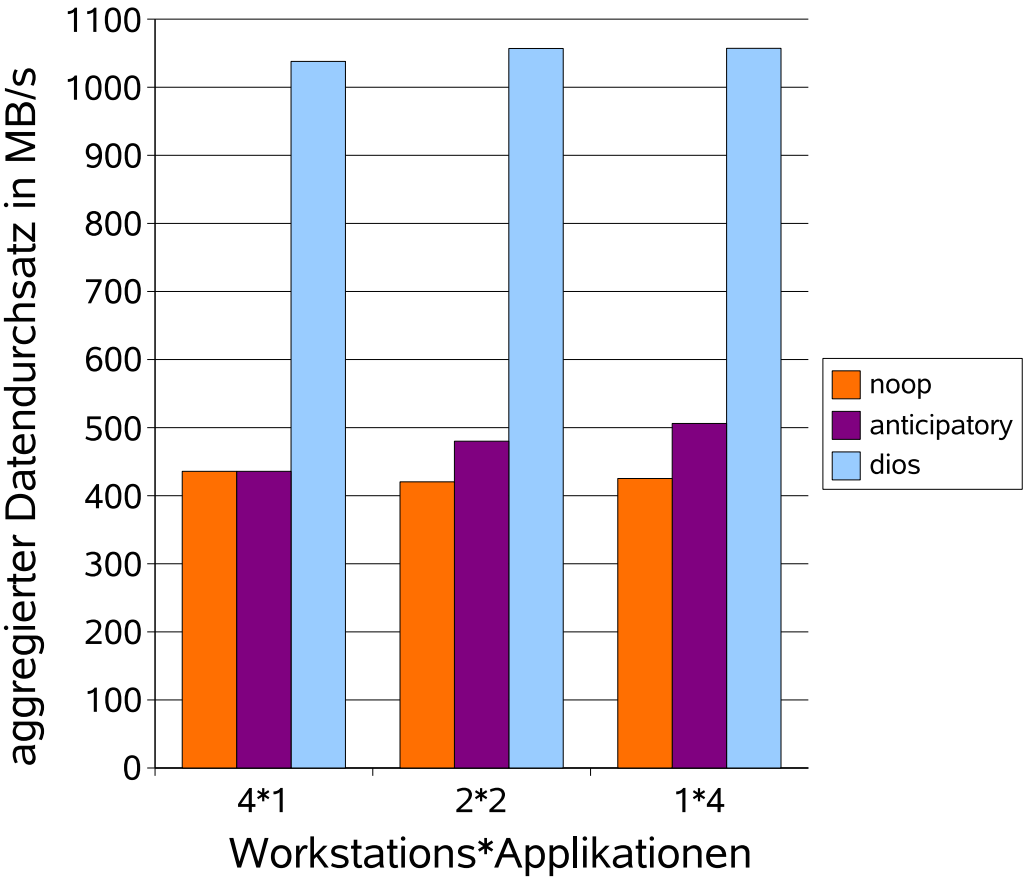

Abb. 14: *Mehrere Applikationen pro Rechner.*

Dazu werden 3 Messreihen durchgeführt. 4 Applikationen auf je einer Workstation (4x1), 2 Applikationen auf 2 Workstations (2x2) und 4 Applikationen auf eine Workstation (1x4).

Wie in Abb. 14 zu sehen, hat es auf die mit DIOS erzielten Datenraten keinen Einfluss, von welchem Rechner die Datenströme gelesen werden. Selbst wenn

alle Ströme auf der gleichen Workstation laufen, ist das DIOS-Framework noch deutlich schneller als der Anticipatory-Scheduler.

6.4.4 Priorisierung von Applikationen

Als nächstes soll überprüft werden, ob eine Applikation, welche Bandbreite beantragt hat, diese auch erhält wenn andere Applikationen extrem hohe Last erzeugen.

Zum Erzeugen der Last wird ein kleines Skript verwendet, welches lediglich Daten vom SAN liest und diese sofort verwirft.

Listing 1: *Listing des Lasterzeugungs-Skriptes*

```
1 #!/bin/zsh
2 while (true) {
3     for i ({001..999}) {
4         dd if=/san/$i.dpx of=/dev/null bs=8388608 count=1
5     }
6 }
```

Während dieses Skript läuft wird nun eine Messreihe mit Sanbs im sogenannten *Constant-Rate-Mode* gestartet. In diesem Modus liest Sanbs mit einer möglichst konstanten Geschwindigkeit, in diesem Fall sollen pro Sekunde 24 Frames zu jeweils knapp 8 MB gelesen werden. Dies entspricht dem Verhalten einer Anwendung die einen HD-Filmstream abspielt. Gemessen wird über einen Zeitraum von 5 Minuten. Die dabei entstehende Datenrate beträgt ungefähr 195 MB/s.

Anzahl Clients	Datenraten mit Last	Datenraten ohne Last
2	195.82	195.71
	195.70	195.66
3	195.53	195.37
	195.70	195.67
	195.23	195.36
4	195.72	196.45
	196.11	195.54
	195.84	196.16
	196.37	196.05

Tab. 3: *Messergebnisse mit dem DIOS-Framework*

Diese Messungen werden mit mehreren Instanzen von Sanbs durchgeführt, jeweils mit und ohne das Lastskript und sowohl mit DIOS als auch mit dem *noop-Scheduler*. Im Falle des DIOS-Frameworks läuft die Applikation im Realtime-Modus, das heißt die benötigte Datenrate soll ihm garantiert werden.

In Tabelle 4 wird deutlich, dass der *noop-Scheduler* aufgrund der Last nicht in der Lage ist einer einzigen die benötigte Datenrate zur Verfügung zu steen, obwohl diese weniger als 20% der physikalischen Bandbreite entspricht. Mit dem DIOS-Framework (Tabelle 3) hingegen werden die garantierten Bandbreiten auch erreicht, unabhängig von der übrigen Last auf dem SAN.

Anzahl Clients	Datenraten mit Last	Datenraten ohne Last
1	151.98	195.85
2	112.07	195.84
	112.06	195.85
3	94.37	142.92
	94.38	142.94
	94.38	142.93

Tab. 4: *Messergebnisse mit dem NOOP-Scheduler*

7 Fazit

Es wurde ein verteilter Scheduling-Mechanismus für Storage Area Networks auf Basis des von Thomson entwickelten DIOS-Frameworks implementiert. Das Problem, der verteilten Zugriffe von mehreren Rechnern aus wurde durch Koordination der Scheduler auf den einzelnen Rechnern gelöst.

Das DIOS-Framework wurde dafür um eine weitere Hierarchieebene erweitert. Während der Server einen Scheduling-Mechanismus für die einzelnen Rechner umsetzt, tut der Client dies für die Applikationen der Rechner. Der lokale Scheduler sortiert die Anfragen der jeweiligen Applikationen in jeweils eigene Queues und reicht stets nur die Anfragen aus einer Queue weiter. So erhält die entsprechende Applikation für einen kurzen Zeitraum exklusiven Zugriff auf das SAN und kann, sofern die Zugriffe sequentiell sind mit der maximal möglichen Datenrate zugreifen. Dies ist optimal auf digitale Postproduktionsumgebungen der Filmindustrie zugeschnitten, da hier fast ausschliesslich mit sequentiell gespeicherten Datenströmen gearbeitet wird. Die Performancegewinne bei mehreren sequentiell gelesenen oder geschriebenen Datenströmen gegenüber den in den Linux-Kernel integrierten Schedulern liegt bei bis zu 100%.

Desweiteren kann für Applikationen mit Echtzeitanforderungen eine beliebige Bandbreite reserviert werden, die dieser dann garantiert kontinuierlich zur Verfügung steht, unabhängig von der übrigen Auslastung des Speichergerätes.

Es wurde eine API entwickelt, über die sich Applikationen beim Scheduler registrieren können um in den Scheduling-Mechanismus priorisiert einbezogen zu werden. Außerdem kann über die API eine konstante Datenrate beantragt werden.

Das DIOS-Framework wurde zu einem voll funktionsfähigem Scheduling-Framework erweitert, welches ohne großen Aufwand in bestehende Storage Area Networks integriert werden kann. Der Server des DIOS-Frameworks wurde neu implementiert, da die vorherige aus zwei Kernel-Modulen bestehende Implementierung fehleranfällig und nur schwer erweiterbar war. Die neue Implementierung ist eine sehr objektorientierte multithreaded Userspace-Anwendung.

Das Framework wurde ausgiebig getestet wodurch festgestellt wurde dass alle Anforderungen erfüllt werden.

Das Framework steht nun kurz vor der Produktionsreife. Durch dessen Einsatz können Filmstudios die Hardware in ihren Postproduktionsumgebungen wesentlich effizienter nutzen und so ernorme Kosten einsparen.

8 Ausblick

Das DIOS-Framework beinhaltet einen eigenständigen I/O-Scheduler, der die vom Betriebssystem mitgelieferten I/O-Scheduler nicht ergänzt sondern vollständig ersetzt. Unter Umständen wäre es sinnvoll, die Funktionalität eines oder mehrere lokalen Scheduler in DIOS zu integrieren. So könnten auch Best-Effort-Prozesse eine höhere Performance erzielen. Das ganze Framework wäre dadurch flexibler und der mögliche Einsatzbereich wäre größer. In dem genannten Szenario der Postproduktion würde dies jedoch sehr wenig oder gar keinen Gewinn bringen, da hier fast ausschließlich sequentiell zugegriffen wird.

Eine Schwierigkeit bei der Inbetriebnahme des Frameworks ist das Festlegen einer guten Zyklusdauer. Eine zu hohe Zyklusdauer erhöht die Latenz, eine zu niedrige würde die Performance erheblich verschlechtern. Ein naheliegender Gedanke ist, die Zyklusdauer in Abhängigkeit der Anzahl der aktiven Applikationen und deren Zugriffsverhalten dynamisch festzulegen und bei Bedarf zu ändern. Dies würde ein weiteres Studieren des Verhaltens der Applikationen voraussetzen.

Das DIOS-Framework wurde ausschließlich unter Linux entwickelt und getestet. Eine Portierung auf andere Betriebssysteme wäre jedoch denkbar. Der DIOS-Server könnte, da er im Userspace läuft und nicht auf Betriebssystemspezifische Funktionen angewiesen ist leicht auf andere - auch proprietäre - Betriebssysteme portiert werden. Bei den lokalen Teile des Schedulers wäre das nicht möglich, da diese sehr tief im Linux-Kernel arbeiten und dessen Schnittstellen nicht zu anderen Systemen kompatibel sind. Es müssten also große Teile neu implementiert werden. Außerdem ist es dazu notwendig, dass die Quellen des Betriebssystem zur Verfügung stehen. Eine Portierung zum Beispiel nach Windows oder Mac OS wäre ohne Zusammenarbeit mit dem Hersteller nicht möglich.

Eine komplexe aber hilfreiche Erweiterung könnte sein, das Verhalten der Applikationen zu analysieren und diese danach selbstständig in Klassen zu unterteilen. Der Scheduler müsste dafür selbstständig erkennen, ob eine Applikation sequentiell zugreift und diese dann einer priorisierten Klasse zuordnen. Auch die Datenrate könnte gemessen und - falls sie konstant ist - der Applikationen eine feste Datenrate zugewiesen werden. Allerdings würde der Anwender so erhebliche Kontrolle über das gesamte Framework verlieren.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[Dis04] Discreet. High performance collaboration. *Stone shared Whitepaper*, November 2004.

[Fü07] Lars Fürst. *Modellierung und Optimierung von verteilten Systemen für heterogene High-Performance-Image-Processing-Anwendungen.* Universität Dortmund, Robots Research Institute, 2007.

[Lov05] Robert Love. *Linux Kernel Development.* Novell Press, 800 East 96th Street, Indianapolis, Indiana 46240 USA, 2nd edition, 2005.

[QK05] Jürgen Quade and Eva-Katharina Kunst. Kern-technik, folge 19. *Linux-Magazin*, März 2005.

[QK06] Jürgen Quade and Eva-Katharina Kunst. *Linux-Treiber entwickeln.* dpunkt.verlag GmbH, 69115 Heidelberg, 2nd edition, 2006.

[Son06] Matthias Sonntag. *Modellierung und Optimierung eines heterogenen Rechner- und Speichernetzes zur Echtzeitbearbeitung von Video in Digitalkinoqualität.* Diplomarbeit Universität Dortmund, 2006.

[TE03] Ulf Troppens and Rainer Erkens. *Speichernetze.* dpunkt.verlag GmbH, 69115 Heidelberg, 1 edition, 2003.

Lizenz

Das bedeutet:

Sie dürfen:

das Werk vervielfältigen, verbreiten und öffentlich zugänglich machen

Bearbeitungen des Werkes anfertigen

Zu den folgenden Bedingungen:

Namensnennung. Sie müssen den Namen des Autors/Rechteinhabers in der von ihm festgelegten Weise nennen (wodurch aber nicht der Eindruck entstehen darf, Sie oder die Nutzung des Werkes durch Sie würden entlohnt).

Keine kommerzielle Nutzung. Dieses Werk darf nicht für kommerzielle Zwecke verwendet werden.

Weitergabe unter gleichen Bedingungen. Wenn Sie dieses Werk bearbeiten oder in anderer Weise umgestalten, verändern oder als Grundlage für ein anderes Werk verwenden, dürfen Sie das neu entstandene Werk nur unter Verwendung von Lizenzbedingungen weitergeben, die mit denen dieses Lizenzvertrages identisch oder vergleichbar sind.

Um die Lizenz anzusehen, gehen Sie bitte zu
http://creativecommons.org/licenses/by-nc-sa/2.0/de/
oder schicken Sie einen Brief an Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.