Fachhochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich Informatik
Department of Computer Science

Seminararbeit

im Master Studiengang – Sommersemester 2005
Veranstaltung: Parallel Systems

Optimierungsmöglichkeiten für PVFS I

Autoren: Iek Leng Lau
Philipp Wever

Seminarleiter: Prof. Dr. Rudolf Berrendorf

Eingereicht am: 18. Juni 2005

Inhaltsverzeichnis

Inhaltsverzeichnis II
Abbildungsverzeichnis III
Abkürzungsverzeichnis IV
1 Einleitung 1
2 Vereinheitlichung von Cache- und Communication-Buffer-Management 2
2.1 Überblick über PVFS und InfiniBand 3
2.1.1 Datentransfer über TCP/IP 5
2.1.2 Datentransfer über InfiniBand 6
2.2 Design von Unifier 7
2.2.1 Grundlegende Architektur 8
2.2.2 Features in der Unifier API (Application Programming Interface) 9
2.2.3 Potenzielle Vorteile 11
2.2.4 Weitere wichtige Eigenschaften 11
2.3 Vorgehensweise bei der Implementierung 14
2.4 Ergebnisse und Auswertungen 14
3 Unterstützung von nicht zusammenhängendem Zugriff in PVFS I 17
3.1 Einführung über nicht zusammenhängenden Zugriff 17
3.2 PVFS list I/O 18
3.3 Mechanismen für verteilten Dateitransfer 20
3.3.1 Multiple Message 21
3.3.2 Pack/Unpack 21
3.3.3 Gather/Scatter 21
3.3.4 Probleme und Vergleich 22
3.4 Effiziente Speicherregistrierung 23
3.4.1 Anwendungsgesteuerte Speicherregistrierung 23
3.4.2 Optimistic Group Registration (OGR) 23
3.4.3 Vorgehensweise bei der Implementierung von OGR 23
3.5 Ergebnisse und Auswertung 23
4 Zusammenfassung und Ausblick 23
Literaturverzeichnis & weiterführende Literatur XXIII

Abbildungsverzeichnis

Abbildung 1 Cluster mit Metadata-Server und Clients [VFS 2002] 4
Abbildung 2 grundlegende Architektur von Unifier..... 9
Abbildung 3 Durchflüsse verschiedenen Subsysteme [Wu 2003]..... 15
Abbildung 4 Cached read Bandbreite 16
Abbildung 5 Effekte von Cachegröße 17
Abbildung 6 PVFS cached read Leistung 17
Abbildung 7: Formen nichtzusammenhängenden Datenzugriffs [Ching 2002] 18
Abbildung 8: PVFS list I/O Interface 19
Abbildung 9: Datentransfer über PVFS list I/O [Wu 2003] 20
Abbildung 10: Multiple Message [Wu 2003]..... 21
Abbildung 11: Pack/Unpack [Wu 2003] 21
Abbildung 12: Gather/Scatter [Wu 2003] 22
Abbildung 13: Gruppierung von Puffern mit OGR (in Anlehnung an [Wu 2003]) .. 23

Abkürzungsverzeichnis

API	Application Programming Interface
CPU	central processing unit
Daemon	Disk and execution monitor
GByte	Gigabyte
IBA	InfiniBand Architecture
I/O	Eingabe/Ausgabe
iod	I/O Server
KByte	Kilobyte
libpvfs	PVFS native API
Mbit	Megabit
MByte	Megabyte
mgr	Metadaten Server
MPI	Message Passing Interface
NFS	Network File System
PVFS	Parallel Virtual File System
RAID	Redundant Array of Independent Disks
RDMA	remote direct memory access
SDK	software development kit
TCP	Transmission Control Protocol
US	United States
VFS	Virtual File System
VI	Virtual Interface

1 Einleitung

Netzwerkspeicherungssysteme (network storage systems) werden immer mehr zu einer Hauptlösung für die I/O (Eingabe/Ausgabe) intensiver Anwendungen in verschiedenen Gebieten, wie Datenzentren, Leistungsrechensystemen, und die korporativen Rechenumgebungen. Netzwerkspeicherungssysteme stellen Potentiale zur Verfügung, um hohe Leistung, Skalierbarkeit, Zuverlässigkeit, und Handlichkeit zu erreichen. Jedoch wird die Leistung von Netzwerkspeicherungssystemen häufig durch die niedrige Leistung des Netzwerksubsystems beschränkt. Die Verschiedenheit zwischen der I/O-Performance und Verarbeitungsperformance führt bei vielen Anwendungen zu I/O-Engpässen, insbesondere bei denjenigen, die große Datensätze verwenden.

Eine populäre Verfahrensweise, dieser Art von Engpässen zu begegnen, ist der Gebrauch von „Parallel Virtual File Systems“ [PVFS 2005], welches in der Seminararbeit „Parallel Virtual File System (PVFS) I - Architektur, Verwendung und Eigenschaften “ vom Christian Staron und Björn Adam ausführlich erläutert wurde.

Einige darin erwähnten Nachteile, bzw. optimierungsbedürftige Stellen von PVFS I stellen die Schwerpunkte dieser Seminararbeit dar.

In Anlehnung an die Zusammenarbeiten zwischen „Computer and Information Science“ von der Staatsuniversität Ohio; „Ohio Supercomputer Center“ aus Columbus und „Mathematics and Computer Science Division“ von „Argonne National Laboratory“ aus Argonne der Vereinigten Staaten, werde einigen Optimierungen von PVFS Version I vorgestellt.

Im Kapitel 2 wird auf eine Komponente, genannt Unifier fokussiert, welche eine effizientere Integration und bessere Interaktion zwischen den Komponenten bewirkt. Das Design, die grundlegende Softwarearchitektur, das Interface und die potenziellen Vorteile des Unifiers werden in Detail erläutert. Des Weiteren wird eine Evaluierung, welche auf den experimentalen Ergebnissen gestützt ist gezeigt.

Kapitel 3 befasst sich mit der Unterstützung nicht zusammenhängenden Datenzugriffs in PVFS I über InfiniBand. Es wird dazu eine kurze Einführung über die Problemstellung nicht zusammenhängenden Datenzugriffs gegeben. Ferner werden Performance-Engpässe bei verschiedenen Mechanismen für den verteilten Dateitransfer aufgezeigt. Im Rahmen der Optimierungsmöglichkeiten wird ein Schema zur effizienten Speicherregistrierung und damit verbundenen Performan-

ceverbesserung eingeführt. Zum Abschluss des Kapitels erfolgt eine Auswertung des vorgestellten Schemas anhand eines Benchmark-Tests.
Abschließend im Kapitel 4 wird eine kurze Zusammenfassung und einen Ausblick gegeben.

2 Vereinheitlichung von Cache- und Communication-Buffer-Management

Die vernetzten Technologien und die hohen Leistungstransportprotokolle erleichtern zwar den Speicherdienst (service of storage) über Netzwerke, stellen jedoch gleichsam Herausforderungen an die Integration und Wechselwirkung unter Speicherserver-Anwendungskomponenten (storage server application components) und Systemkomponenten.
Netzwerkarchitekturen wie Virtual Interface (VI) Architektur und InfiniBand Architektur (IBA) stellen als zwei Hauptmerkmale “user-level networking” und “remote direct memory access (RDMA)” zur Verfügung, welche niedrige Zugriffzeit, hohen Datendurchfluss, und niedrige CPU Overhead-Kommunikation in Netzwerkspeicherungssystemen ermöglichen.
InfiniBand ist eine neue Architektur zur Verbindung von Prozessorknoten und Ein-/Ausgabeknoten, die sich besonders für Rechner-Cluster eignet. Im Vergleich zu existierenden Technologien bietet ein InfiniBand-Netzwerk hohe Bandbreiten und geringe Latenzzeiten [InfiniBand 2005]
Unter RDMA ist ein direkter Speicherfernzugriff zu verstehen, der für den Zugriff auf externe Geräte über Netzwerk und die Serververnetzung in Rechenzentren auf Basis von 10-Gigabit-Ethernet entwickelt wurde.
Diese Technologien beseitigen, bzw. reduzieren Kosten der Speicherkopie, vernetzen Zugang, Unterbrechung und Protokollbearbeitung im Netzsubsystem. Jedoch gibt es noch eine Menge von Herausforderungen zu bewältigen. Eines der bedeutendsten Probleme ist effizientes „Communication-Buffer-Management“, um Memoryregistrations- und -deregistrationsaufwand zu reduzieren.
Eine andere Quelle der Leistungsbeschränkung in Netzwerkspeicherungssystemen ist der Mangel an der Integration innerhalb verschiedener Systemkomponenten (z.B. der Dateicache, das Dateisystem und das Netzwerksubsystem) und die Speicherserveranwendungen (storage server applications) im Mehrzweckbetriebssystem (general-purpose operating system). Das verursacht oft redundanten

Datenkopien, multiple buffering und anderen Leistungsminderungen (performance degradation).

- Redundante Memory-Kopien (memory copying) führen zu hohen CPU Overhead und reduzieren damit den Datendurchfluss des Servers. In Netzwerken wie IBA, die vergleichbare Leistung zum Speichersystem zur Verfügung stellt, kann das der primäre Leistungsengpass sein.
- Multiple buffering von Daten verschwendet Memory. Infolgedessen wird die effektive Cachegröße reduziert und der Plattenzugriff erhöht.

Insgesamt betrachtet verhindert die verengte Schnittstelle zwischen Systemkomponenten und Anwendungen eine effiziente Kooperation.
In folgenden wird die Lösung für diese Probleme vorgestellt.

2.1 Überblick über PVFS und InfiniBand

Um eine Vollständigkeit bzw. das Verständnis zu gewährleisten, wird zunächst ein kurzer Überblick über PVFS und InfiniBand gegeben.
Parallel Virtual File System (PVFS) ist ein führendes paralleles Filesystem für Linux Cluster System. PVFS verbindet per Netzwerk verschiedene Partitionen auf verschiedenen Rechner zu einer logischen Gesamtpartition. Das geschieht durch Abbilden dieser Partitionen auf den Metadata-Server. Ein PVFS Filesystem wird durch ein auf einem Metadata-Server (HEAD) laufenden Daemon (Disk and execution monitor) und durch die auf den sonstigen (DATA) Rechner (n1-nX) laufenden Daemonen (iod, pvfsd), sowie einen Kernel-Modul (pvfs.o) realisiert. Eine Schematische Konfiguration für einen Cluster mit einem Metadata-Server und 8 Data-Rechnern könnte graphisch folgendermaßen dargestellt werden [VFS 2002]:

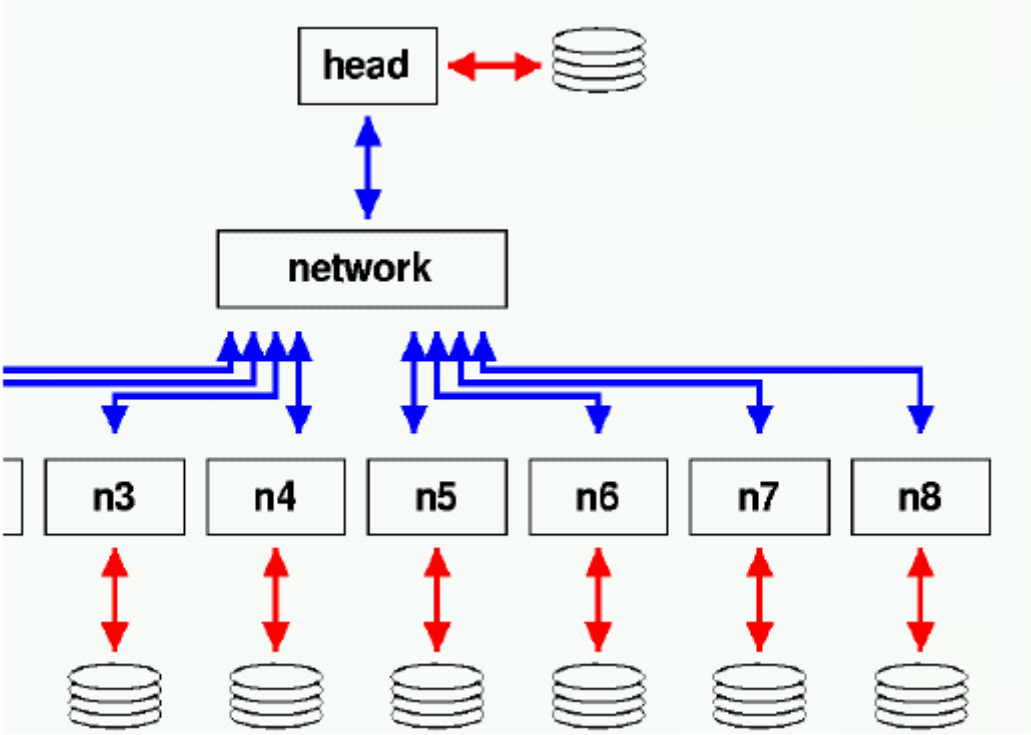

Abbildung 1 Cluster mit Metadata-Server und Clients [VFS 2002]

PVFS erreicht hohe Leistung durch Aufteilen der Dateien über eine Reihe von I/O-Server-Knoten, um parallele Zugänge und aggregierte Leistung zu erreichen. Ein I/O-Daemon läuft auf jedem I/O-Knoten und den Serviceanfragen (Lese- und Schreibanforderungen) von den Rechen-Knoten. Somit werden die Daten direkt zwischen I/O-Servern und Rechen-Knoten transferiert. PVFS verwendet das native Filesystem auf den I/O-Servern, um die individuelle Datenaufteilung abzuspeichern.

Ein Manager-Daemon läuft auf einem Metadata-Manager-Knoten. Es behandelt Metadata-Operationen (z.B. die Dateierlaubnis, Abbruch, Dateieigenschaften der Zerlegung etc.). Metadata wird auch im lokalen Dateisystem abgelegt. In PVFS nimmt der Metadata-Manager an Lesen/Schreiben-Operationen nicht teil.

Zusammengefasst stellt PVFS die folgenden Eigenschaften zur Verfügung:

- Kompatible mit der vorhandenen Dualzahlen
- Leicht zu installieren
- Benutzerkontrollierte Aufteilung der Dateien über Knoten
- Vielfache Schnittstellen, einschließlich einer MPI-IO-Schnittstelle über ROMIO
- Verwertet Warennetz und Speicherungshardware [PVFS 2005]

Wie zum Beginn dieses Abschnitts bereits kurz erklärt: InfiniBand ist ein serieller Bus mit sehr hoher Geschwindigkeit zum internen und externen Einsatz. InfiniBand benutzt einen bidirektionalen seriellen Bus zur kostengünstigen und latenzarmen Datenübertragung. Trotzdem ist es sehr schnell und schafft Datenübertragungsraten bis zu 10 GBit/s in beide Richtungen.
Die Einsatzgebiete von InfiniBand reichen von Bussystemen bis zu Netzwerkverbindungen. Es wird aktuell nur als Cluster-Verbindungstechnologie benutzt. Der große Vorteil von InfiniBand gegenüber gebräuchlichen Technologien wie TCP/IP/Ethernet liegt dabei in der Minimierung der Latenzzeit durch Auslagern des Protokollstacks in der Netzwerkhardware. [Wikimedia 2005]
Um die Zugriffszeiten weiter zu minimieren, stellt InfiniBand zwei Verbindungsmodi zur Verfügung, welche Daten in den Hauptspeicher eines anderen Knotens übertragen oder von dort lesen, ohne das Betriebssystem oder den Prozess auf der Gegenseite zu involvieren. Diese beiden Operationen werden als RDMA Write/RDMA Read bezeichnet.
Eine InfiniBand Architektur definiert ein System-Bereichsnetzwerk (System Area Network), um sowohl Processing-Knoten als auch Eingabe/Ausgabe-Knoten miteinander zu verbinden. Sowohl Kanal- als auch Memory-Semantik sind für den Datentranfer nutzbar. In der Kanal-Semantik werden send/receive Operationen für die Kommunikation verwendet. In der Memory-Semantik werden RDMA wirte/RAMDA read Operationen verwendet. Ein Grunderfordernis in dem derzeitigen „software development kit“ (SDK) von InfiniBand ist, dass der Datenbuffer vor jeglicher Kommunikation registriert sein soll [InfiniBand 2005] [Wu 2003].

2.1.1 Datentransfer über TCP/IP

Zum Grundverständnis für die späterer Ausführungen wird zunächst die Datentransfermethode über TCP/IP erläutert.
Der Eingabe/Ausgabe-Pfad in einem PVFS Eingabe/Ausgabe-Server verbindet sowohl Netzwerk-Eingabe/-Ausgabe-Operationen als auch Datei-Eingabe/Ausgabe-Operationen. Deshalb hängt die Effizienz des PVFS Eingabe/Ausgabe-Servers von der Leistung dieser beiden Operationen, sowie von der Wechselwirkung zwischen ihren verbundenen Subsystemen ab: das Netzwerksubsystem und das Dateisystem. In der Implementierung von PVFS über TCP/IP werden drei Datentransfer-Methoden unterstützt:

- In der Normal-Methode übersetzt ein PVFS-Server eine PVFS Leseanfrage in zwei getrennten Aufrufen: ein Leseaufruf der Datei und ein Schreibaufruf des Netzwerks (a file read call and a network write call). Ähnlich verläuft dies bei einer Schreibanfrage, welche in einem Leseaufruf des Netzwerks und einem Schreibaufruf der Datei übersetzt wird. Hierfür werden normalerweise vier Kontextswitches benötigt und mindestens zwei Datenkopien zwischen User Buffer und File Cache, bzw. User Buffer und Netzwerk Buffer gegeben.
- Die Mmap-Methode benutzt den Systemaufruf „mmap“, um den angeforderten Teil einer Datei in dem Anwendungsbenutzerraum (application user space) abzubilden. Die Anwendung Lesen oder Schreiben über den abgebildeten Puffer erfolgt in eine Dateilesen oder -schreiben. So werden Datenkopien zwischen dem User Buffer und dem File Cache vermieden.
- „Sendfile“ ist ein Systemaufruf, der direkten Datentransfer zwischen zwei Datei-Descriptors bereitstellt. Mit Sendfile kann ein PVFS-Server Datei-Lesen und Netzwerk-Schreiben in einem Aufruf erledigen. Das reduziert nicht nur Kontextswitches, sondern spart auch zwei Datenkopien. Über Netzwerken mit „Zero-copy TCP/IP“ Implementation ermöglicht die Sendfile-Methode den Zero-copy I/O Pfad für die Datenübertragung von Dateien zum Netzwerk [ZeroCopy 2005].

2.1.2 Datentransfer über InfiniBand

Laut Testergebnis der Arbeit „Unifing Cache Management and Communication Buffer Management“, ermöglicht das PVFS über die InfiniBand native Transportschicht in demselben IBA-Netz eine dreifache Verbesserung im Vergleich zum PVFS über TCP/IP, wenn die Leistung des lokalen Dateisystems neben Netzwerksystem gut balanciert ist. Die Normal- und Mmap-Methoden können auf PVFS über InfiniBand angewandt werden, wenn die InfiniBand native-Transportschicht verwendet wird. Die Sendfile-Methode hingegen kann nicht direkt verwendet werden, weil Sendfile-Methode in der recvfile-Semantik nicht unterstütz wird.

Insgesamt betrachtet gibt es noch mehrere Probleme, die gezielt verbessert werden müssen:

- Datenkopien zwischen verschiedenen Komponenten: I/O-Daten werden zwischen File Cache und PVFS Server Communication Buffer kopiert. Das geschieht, wenn die Normal-Methode verwendet wird oder wenn dynamische Speicherregistrierung und -deregistration in der Mmap-Methode vermieden werden soll. Datenkopien verursachen einen hohen Overhead für PVFS Lese- und Schreiboperationen.
- Explicit communication buffer pool: Es ist eine oft praktizierte Lösung, eine Liste von Puffern vorzuregistrieren und ständig fortzusetzen, um sie für die ganze Kommunikation zu verwenden und aufwändige dynamische Speicherregistrierung und -deregistration zu vermeiden. Um einer großen Zahl von Anfragen gleichzeitig zu bearbeiten, wird ein großer Speicherraum benötigt. Da diese Buffer nicht austauschbar sind, wird die effektive Größe des Hauptspeichers reduziert, und somit auch der File Cache des Servers verkleinert.
- Duplikation der Daten im Communicationsbuffer: wenn ein explicit communication buffer pool verwendet wird, können dieselben Daten in diesen Buffer vorhanden sein, um verschiedenen Anfragen zu bearbeiten. Ein solches Duplikat reduziert die effektive Größe des Communicationbuffers.
- Dynamische Speicherregistration und – deregistration: wenn die Mmap-Methode verwendet wird, könnte die Leistung durch die Speicherregistration und –deregistration um 35% heruntergesetzt werden. Diese Probleme wurden durch den Mangel an Integration und Interaktion unter der PVFS-Transportschicht über InfiniBand verursacht.

Um diese Probleme zu lösen, werden der Communicationsbuffer-Raum und der Cacheraum vereinigt. In dieser Komponente wird ein Application-Level-Cache eingesetzt, so dass der Cacheraum für die Kommunikation direkt verwendet werden kann. Diese Komponente wird Unifier genannt.

2.2 Design von Unifier

In diesem Abschnitt wird der Unifier vorgestellt, wobei das Design, die grundlegende Softwarearchitektur, das Interface und die potenziellen Vorteile im Einzelnen erläutert werden.

Unifier ist ein Bestandteil, welcher der Server-Anwendungen ebenso dient, wie Netzwerkspeicherungssystem-Servern und andere anwendungsspezifischen I/O.

Er versucht, effiziente Interaktion und Integration unter allen Bestandteilen der Serveranwendung zu ermöglichen. Unifier wurde entworfen, um die Leistung von Server-Anwendungen zu verbessern. Er dient gleichzeitig als Cachemanager und Kommunikationsbuffermanager, welche den vorregistrierten Kommunikationsbuffer unterstützen ohne die effektive Cachegröße zu reduzieren. Darüberhinaus bietet Unifier weitere Features, die eine bessere Kooperation mit verbundenen Komponenten ermöglichen.

2.2.1 Grundlegende Architektur

In der folgenden Abbildung werden die grundlegende Architektur vom Unifier und die Interaktionen mit anderen Komponenten veranschaulicht.

Die gestrichelten Linien stellen den Kontrollfluss dar. Unifier, als ein Mittelpunkt, interagiert mit dem Anfragemanager (Request Manager), dem Transport- und dem Speicherkomponenten. Zuerst erhält er eine Anfrage vom Anfragemanager. Dann wird Cache Buffer zur Verfügung gestellt, um diese Anfrage zu bearbeiten. Für eine Leseanfrage fragt der Unifier zuerst die Speicherkomponente ab, ob die angefragten Daten dort bereits zwischengespeichert sind; wenn nicht, werden diese Daten eingelesen. Dann stellt es denselben Buffer dem Transportkomponenten zur Verfügung, um Daten an den Clients zu übersenden. Für eine Schreibanfrage werden die Daten zuerst im Cache Buffer der Transportkomponente gespeichert. Diese Daten werden dann im Cache Buffer vom Unifier zwischengespeichert und zum passenden Zeitpunkte an die Speicherkomponente übertragen.

Die durchgezogenen Linien stellen den Datenfluss dar. Alle Daten werden in den Cache Buffer vom Unifier gelegt, der auch von der Transportkomponente für die Kommunikation, sowie der Speicherkomponente für die Datei- und Speichern-I/O-Operation verwendet wird. Es gibt nur eine Kopie in dem Cache Buffer von Unifier, die von allen Komponenten gemeinsam sicher und gleichzeitig verwendet wird.

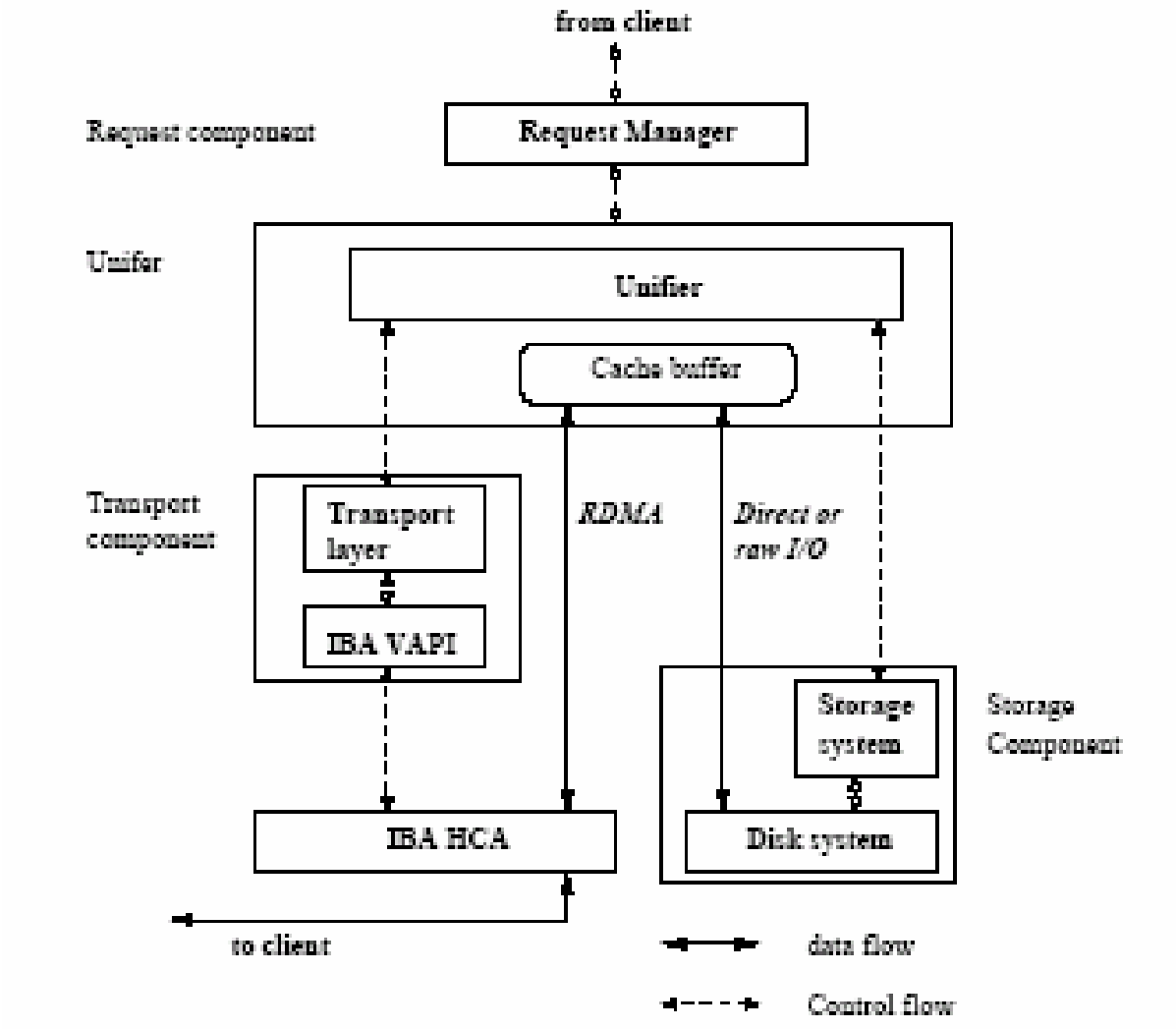

Abbildung 2 grundlegende Architektur von Unifier

Unifier agiert als Cachemanager, der die Application-Level Cache unterstützt und gleichzeitig auch als Buffermanager, der ebenfalls den Buffer für die Transportkomponenten unterstützt.

2.2.2 Features in der Unifier API (Application Programming Interface)

Die Gestaltung des Unifier APIs besteht darin, dass ein Hochleistungs-API das Design von den Hochleistungsserver-Architekturen adaptieren sollte.
Die folgenden Features werden im Unifier API unterstützt:

- Das Unterstützen des strukturierten Datenzugangs: Strukturierter Datenzugang ist ein allgemeines Zugriffsmuster in vielen Anwendungen. Die strukturierte Datenzugriffsunterstützung ist in jedem Bestandteil ein Schlüssel für die hohe Leistung. Die Unifier API sollte diese Voraussetzung erreichen

und mögliche Optimierungen für den strukturierten Datenzugang ermöglichen.

- Das Unterstützen asynchroner Operationen: Asynchrone Operationen stellen eine Möglichkeit dar, I/O-Operationen mit der anderen Prozessen zu verbinden. Netzwerk I/O-Operationen in IBA sind asynchron. Datei und Speicherungssysteme wurden entwickelt, um asynchrone I/O-Unterstützung zur Verfügung zu stellen. Unifier API sollte eine Schnittstelle zur Verfügung stellen, um asynchrone Operationen zu unterstützen und die Fortschritte sowohl im Netzwerk als auch in Speicher I/O auszunutzen.
- Eine expressive Schnittstelle: Bedeutende Forschungsarbeiten haben darauf hingewiesen, dass schmale Schnittstellen in den vorhandenen Systemen eine Barriere für verschiedene Subsysteme geworden sind, um ihre semantische Information auszutauschen, oder die Systemleistung zu verbessern. Es wird eine expressive Schnittstelle erwartet, welche mehr Quersubsystem-Optimierungen und flexibel erweiterte Dienstleistungen erlaubt.

  Das neu definierte Interface beinhaltet fünf Typen von Anrufen: 1) Eine Anfrage stellen; 2) die Vollständigkeit der Anfrage überprüfen; 3) Cache Information anfragen; 4) Vervollständigung anzeigen; 5) Ressourcen freigeben.
  Beispielsweise wurde „Unifier_post_read“ verwendet, um zu zeigen, wie die oben genannten Features erreicht wurden:
  Unifier post read(int fd,
  ACCESS Agg * access info,
  BUFFER Agg * buffer info,
  INFO Agg * semantic info,
  COMP Info * comp info)
  In „Unifier_post_read“ sammelt “ACCESS_Agg” Information von dem strukturierten Zugriff an. Diese gesamte Struktur kann durch einen MPI Datatype leicht vertreten werden, wenn andere Bestandteile Datatype direkt, oder eine Darstellung des strukturierten Zugangs akzeptieren. „INFO_Agg“ beinhaltet semantische Information, welche der Aufrufende zum Unifier durchstellen möchte. „COMP_Info“ führt Unifier, um eine vollständigte Notifikation aufzustellen. Unifier_post_read Opeeration gibt den Buffers zurück, welche

die erfragten Daten beinhalten. „BUFFER_Agg“ sammelt eine Liste von Buffers, welche den Transportbestandteil für die Kommunikation bereitstellen.

2.2.3 Potenzielle Vorteile

Das primäre Ziel von Unifier ist, die Leistung von PVFS I/O-Servern zu verbessern. Es bietet die folgenden potenziellen Vorteile:

- Zero-copy I/O serving: Unifier eliminiert Datenkopien zwischen PVFS Serverkomponenten in dem I/O Pfad. Ferner unterstützt er ein Application-Level Cache, welcher der Speicherkomponente ermöglicht, den Cache der Betriebssystemdaten zu umgehen, ohne dabei Leistung zu verlieren.
- Increased Cache size: Unifier beseitigt die multiple Buffering. Jedes Objekt kann nur eine einzelne Kopie im Cache Buffer des Unifiers haben. Das vergrößert die effektive Größe des Caches, und somit auch die Erfolgsrate (hit rate). In Anbetracht der zunehmenden Spalte zwischen dem Memorysystem und dem Plattensystem, bzw. zwischen dem Netzwerksystem und dem Plattensystem, kann eine kleine Zunahme der Erfolgsrate im Cache die Leistung der I/O intensive Anwendungen bedeutsam verbessern.
- Reduced memory registration and deregistration costs: ein Teil vom Cache Buffer im Unifier kann für die Kommunikation ohne jede Speicherregistration oder –deregistration auf diesen Buffern vorregistriert werden.
- Native structured data access support: Diese Unterstützung des strukturierten Datenzugriffs passt nicht nur zu allgemeinen Anwendungszugriffsmusters gut, sondern stellt ein enormes Optimierungspotential sowohl im Unifier als auch in anderen Bestandteilen dar.

Außer der oben genannten Punkte weist Unifier noch weitere Vorteile auf, welche an diese Stelle nicht näher erläutert werden.

2.2.4 Weitere wichtige Eigenschaften

Unifier und die unifierbasierte I/O-Server-Softwarearchitektur zeigen zwar attraktive potenzielle Vorteile, um höhere Leistung zu erzielen, jedoch müssen auch einige Schwierigkeiten überwunden werden.

- Anpassungsfähiger (adaptiv) PVF I/O server Cache: Applikcation-level Cache wird allgemein in vielen Serverapplikationen, wie zum Beispiel Datenbankmanagement-Applikationen, Webserver-Applikationen und Rasterda-

tenserver (grid data server) angewandt. Der Grund für den Aufbau des PVFS I/O-Server-Caches ist, dass die Anwendungen, die PVFS verwenden, unterschiedliche I/O-Auslastungscharakteristiken und I/O-Anforderungen zu den anderen Systemen haben. Verglichen mit den Datenbankapplikationen haben die PVFS-Applikationen abwechselungsreicheres Zugriffsmuster, aber andererseits wiederum weniger Variationen im Zugriffsmuster als die Mehrzwecksysteme (general-purpose systems). Der Aufbau vom PVFS-I/O-Server soll diesen Unterschied widerspiegeln und eine hohe Leistung im Allgemeinen unterstützen. Ein anpassungsfähiger Cache wird erwartet, um unterschiedlichste Anforderungen zu decken.

Um dies zu erreichen wird die Cacheinformation explizit zu anderen Komponenten ausgestellt (expose). Forschungsarbeiten zeigen, dass die Anwendungen ihr eigenes Verhaltens an das des Operationssystems (OS) anpassen können, um eine verbesserte Leistung mit der Cacheinformation zu erreichen. Unifier stellt ausführliche Informationsfragen des Caches zur Verfügung, um Anpassung zu ermöglichen.

Die Anwendungen können die Cache-Anforderungen näher bestimmen. Diese Informationen werden dem Unifier geliefert. Folglich können verschiedene Cachepolicen angewandt und verschiedene Cacheeinheiten genutzt werden.

- Buffer Sharing: im Unifier teilen Netzwerk-read/-write und Datei/Speicher-read/-write eine einzelne Dateikopie. Das läuft auf Probleme der Synchronisation und Konsistenz im Buffer Sharing hinaus. Techniken wie „immutable Buffer“ (unveränderliche Puffer), die in IO-Lite verwendet werden, können verwendet werden, um diese Probleme zu beheben. „Immutable Buffer“unterstützt „read only“-Buffer Sharing, um Probleme der Synchronisation und Konsistenz zu eliminieren. Der Nachteil hierbei ist, dass Daten nicht allgemein modifiziert werden können. Aus diesem Grund ist der „immutable Buffer“ für wissenschaftliche Anwendungen nicht passend, wo Modifizierung unabdinglich ist.

  Nachdem die wissenschaftlichen Anwendungen das Hauptziel vom PVFS sind, muss eine andere Lösung für Buffer Sharing verwendet werden. Das Modell „allocate-release“ wird eingesetzt, um die Teilung des Cache Buffer zu managen und kontrollieren.

Die Hauptdesignpunkte sind folgende:

Single owner: der Unifier ist der einzige Inhaber von allen Cache Buffers. Das bedeutet, dass Unifier alle Buffer Sharing kontrolliert. Diese Methode reduziert die Aufbaukomplexität bedeutend.

Allocate: Unifier teilt die Cache Buffer den einzelnen Operation zu. Wenn ein Sharing-Konflikt entsteht, wird die Zuteilung verschoben. Wenn kein Konflikt vorhanden ist, darf der gleiche Cache Buffer an verschiedenen gleichzeitig laufenden Operationen zugeteilt werden. Das ermöglicht den sicheren und gleichzeitigen Gemeinnutzen.

Release: wenn eine Operation mit Cache Buffer gestattet ist, soll sie diesen Buffer an dem Unifier wieder freigeben, wenn sie beendet ist. Damit unterstützt Unifier sowhl „read-only“-Sharing, als auch den „write“-Sharing. I/O-Daten können modifiziert werden, wenn sie nicht gerade verwendet werden. Hierfür unterstütz Unifier die „Sendfile“-Semantik über InfiniBand Transportprotokolle, welche die Daten vom Cache Buffer -ohne irgendeine Kopie- direkt ins Netzwerk überträgt.

- The Size of Registered Cache Buffers: ein anderes Hauptziel vom Unifier ist den Memoryregistration/-deregistrationsaufwand bei der RDMA-Operationen zu reduzieren. Um dies zu erreichen, soll die Cachegröße des Unifiers so groß wie möglich sein. Alle freien Memorys werden als Cache verwendet, um die Erfolgsrate zu erhöhen.

  Des Weiteren sollen während der Cacheinitialisierung so viele Cache Buffer wie möglich registriert sein, wobei die Größe der eingetragenen Cache Buffer beschränkt sein soll, damit die Systemleistung nicht beeinträchtigt wird. Die registrierte Buffers sind festgelegt und nicht austauschbar. Die effiziente Größe des physikalischen Memorys, der auch für andere Zwecke verwendet wird, wird dadurch reduziert. Hierfür werden Cache Buffers in zwei Gruppen geteilt: „Ready“-Buffers sind registriert und bleiben während der Lebenszeit des Unifiers im Sytem. „Raw“-Buffer sind während der Cacheinitialisierung zugeteilt, werden jedoch nicht registriert. Die Kommunikation auf diesen Buffers braucht „on-the-fly“-Registration und Deregistration.

  Hinsichtlich des geschätzten benötigten Speichers bei einer PVFS Server-Anwendung und dessen maximale Unterstützung bei außergewöhnlichen Anfragen wird die Größe des „Ready“-Buffers konservativ geplant. Die Grö-

ße von „Raw“-Puffern ist die physische Totalspeichergröße abzüglich der Größe der „Ready“-Puffer und der Größe des Memorys die von einer PVFS Server-Anwendung mit einer leichten Auslastung gebraucht werden.
Mit diesem Design kann eine gute Wechselwirkung zwischen dem Aufwand der Memoryregistrierung/-deregistration und dem Aufwand der potenziellen virtuellen Speichertätigkeiten gewährleistet werden.

2.3 Vorgehensweise bei der Implementierung

In diesem Abschnitt wird eine Übersicht über die Implementation der Unifierkomponenten und deren Entwicklung in PVFS über InfiniBand gegeben.
Unifier wurde als eine User-Level-Komponente in der PVFS Softwarearchitektur implementiert. Als Prototyp basiert die Implementierung des Caches größtenteils auf der Datei Cache Implementation in Linux 2.6. Diese Implementation unterstützt Cache Einheitsgrößen von 4 Kbytes bis 64 Kbytes.
Anwendungen können Unifier empfehlen, eine Cache Einheitsgröße für eine Datei zu wählen, wenn die Datei zuerst geöffnet wird. Unifier verwendet den O_DIREKTE, um Lesen und Scheiben einer Datei zu unterstützen, womit der Systemcache umgangen wird.
Der strukturierte Datenzugang wird durch Verwenden einer Liste von <offset, length> Paare in der Implementierung unterstützt.
In der gegenwärtigen Implementation stellt Unifier dem Request-Manager ausführliche Informationsfragen zur Verfügung. Wie man jedoch von der Cache Information Gebrauch macht, wird weiter erforscht. An einem anpassungsfähigen Cache Management wird gearbeitet.

2.4 Ergebnisse und Auswertungen

In diesem Abschnitt werden die in der Arbeit „Unifying Cache Management and Communication Buffer Management for PVFS over InfiniBand“ durchgeführten experimentellen Ergebnisse und die Auswertungen vorgestellt.
Die durch PVFS-Anwendungen realisierte Leistung hängt von der Leistung dreier Hauptsubsystemen ab: dem Netzwerk, dem Speichern, und dem Dateisystem. In der folgenden Tabelle werden die Durchflüsse von IBA VAPI Send/Recv, RDMA Write, RDMA Read, Speicherkopie, Datei lesen und schreiben mit und ohne Cache verglichen. Im IBA Durchfluss-Tests werden Speicherregistrierungs- und De-

registrationsaufwand nicht eingeschlossen. Im Speichernkopien-Test, ist die kopierte Datenmenge mit 20 Megabytes, viel größer als L1- und L2-Cache, um die Wirkung des Caches zu vernachlässigen.

Subsystem	Throughput (MB/s)
VAPI Send/Recv	830
VAPI RDMA Write	830
VAPI RDMA Read	826
Memory Copying	596
File Read w/o cache	20
File Write w/o cache	25
File Read w/i cache	590
File Write w/i cache	476

Abbildung 3 Durchflüsse verschiedenen Subsysteme [Wu 2003]

Die folgende Abbildung zeigt, dass die Normal-Mehthode die höchste Bandbreit von 324 MByte/sec aufweist. Wenn die Zugriffsgröße 128 KBytes übersteigen, fällt die Bandbreite ab.

In der Mmap-Methode haben die Memoryregistration und –deregistration eine bedeutsame Auswirkung auf die kleinere Zugriffsgröße. Wenn die Zugriffsgröße zunimmt, wird der Aufwand der Memoryregistration und –deregistration weniger sein, als der Aufwand der Memorykopien.

In der Unifier-Methode werden die Daten im „Ready“-Cache Buffers zwischengespeichert. So kann der Server RDMA write Daten direkt von den Cache Buffern des Unifiers in den Buffer vom Client schreiben.

Unifier erreicht gegenüber der Normal-Methode einen Verbesserungsfaktor von 2,1 und einen Verbesserungsfaktor von 1,3 gegenüber der Mmap-Methode. Wenn die Zugriffsgröße größer wird, steigt der Faktor sogar über 2,7.

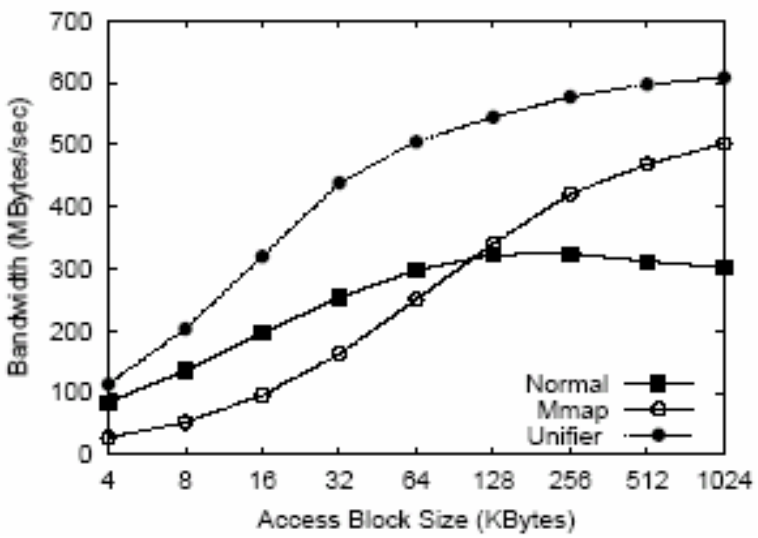

Abbildung 4 Cached read Bandbreite

Die folgende Abbildung veranschaulicht das Testergebnis einer effektiven Cachegröße: eine Datei hat die Größe zwischen 300 MBytes bis 400 MBytes. Diese Datei wurde sequenziell mit einer Blockgröße von 128 KBytes eingelesen. Es ist deutlich zu erkennen, dass Mmap-Methode und Unifier-Methode den Eingang der Datei bis 360 MBytes aufnehmen können, während die Normal-Methode dies nicht schafft.

Wenn die Dateigröße auf 380 Megabytes zunimmt, reduzieren alle Methoden deren Bandbreite auf 20 MBytes/sec, weil der platten-bestimmte Zugang auf eine normalen IDE Platte erfolgt, welche lediglich eine „read“- Bandbreite von 20 MBytes/sec hat. Alle Methoden sind vergleichbar. Das zeigt, dass Unifier Cache vergleichbare Leistungen zum Systemcache mit sequenzieller Auslastung unterstützen kann.

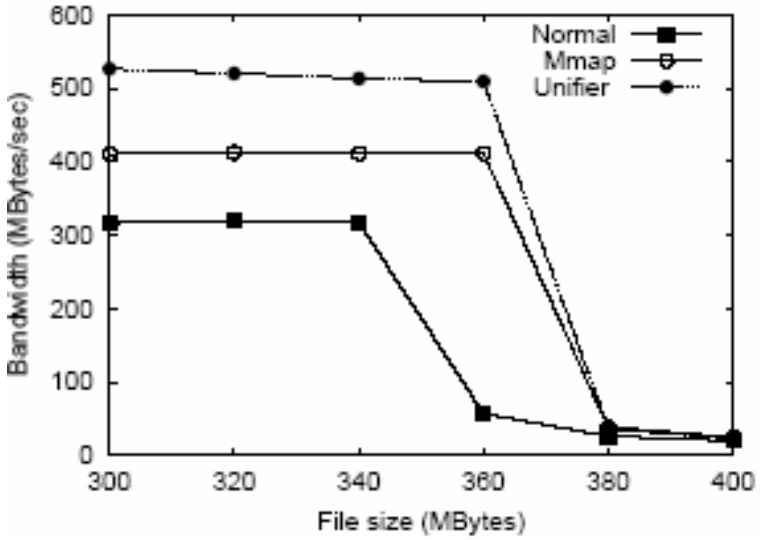

Abbildung 5 Effekte von Cachegröße

Folgende Abbildung zeigt die Gesamtbandbreite bei mehreren Clients. Es ist deutlich zu erkennen, dass PVFS mit Unifier infolge des niedrigeren CPU-Overhead in der Unifier-Methode besser steigt, als mit den anderen Methoden. Beim Spitzenwert der Bandbreite liegt ein Verbesserungsfaktor von 1,7 gegenüber der Normal-Methode und 1,3 gegenüber der Mmap-Methode.

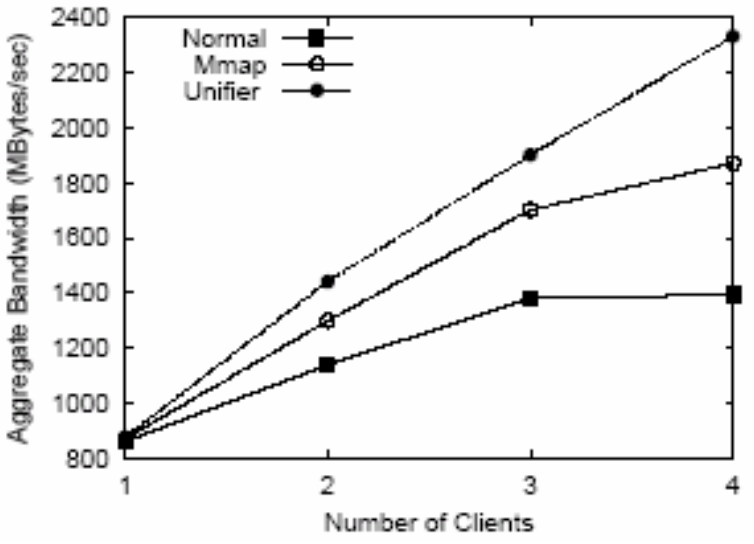

Abbildung 6 PVFS cached read Leistung

3 Unterstützung von nicht zusammenhängendem Zugriff in PVFS I

In heutigen Clustersystemen zählen I/O Vorgänge zu den Hauptgründen für reduzierte Performance. Die Entwicklung von parallelen Dateisystemen hat dabei geholfen diesen Schwachpunkt zu kompensieren. I/O ist jedoch weiterhin ein Gebiet, welches Lösungen zur Performancesteigerung bedarf. Dieses Kapitel stellt Möglichkeiten vor, wie ein nicht zusammenhängender Datentransfer für das erweiterbare Clusterdateisystem PVFS I optimiert werden kann, um die I/O Performance zu optimieren.

3.1 Einführung über nicht zusammenhängenden Zugriff

Nichtzusammenhängender Zugriff befasst sich mit der Problemstellung auf Datenbereiche zugreifen zu müssen, die innerhalb des Speichers oder innerhalb von Dateien nicht benachbart bzw. lückenhaft sind. Verschiedene Typen nicht zusammenhängenden Datenzugriffs sind in **Fehler! Verweisquelle konnte nicht gefunden werden.** zu sehen.

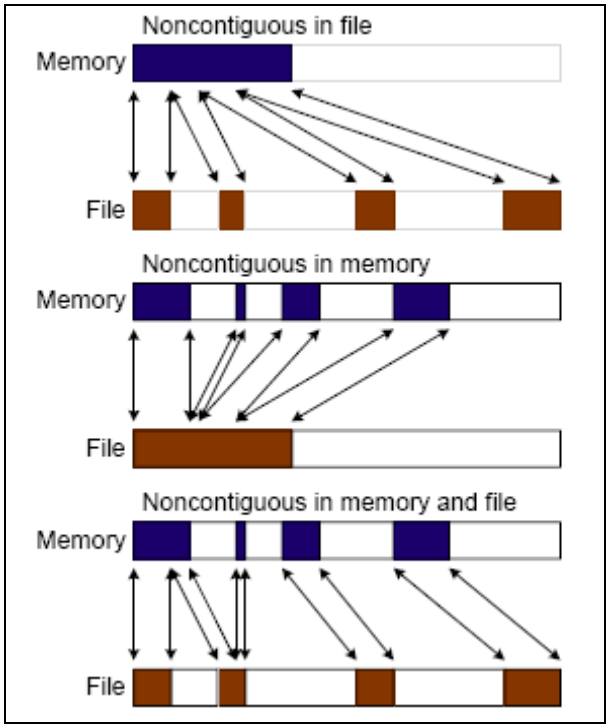

Abbildung 7: Formen nichtzusammenhängenden Datenzugriffs [Ching 2002]

Ein Beispiel für zusammenhängende Daten im Speicher und nicht zusammenhängende Daten in einer Datei ist eine Anwendung, die ein zweidimensionales Array in einer Datei ablegt und später aus jeder Zeile ein Element in einen zusammenhängenden Speicherpuffer lesen möchte.
Sind Daten innerhalb einer Datei zusammenhängend angeordnet, so reicht für den Zugriff eine einzige Schreib-/Leseoperation aus. Wenn eine Datei nicht zusammenhängend aufgebaut ist, müssen andere Zugriffsmethoden verwendet werden, die für jede Dateiregion eine separate I/O Operation erfordern.
Diverse Studien haben ergeben, dass nicht wenige wissenschaftliche Anwendungen auf eine Vielzahl kleiner, nicht zusammenhängender Datenbereiche in Dateien zugreifen. Wenn für diese Vorgänge herkömmliche I/O Anfragen, die von zusammenhängenden Daten ausgehen, angewandt werden, hat dies große Performanceverluste zur Folge. [Ching 2002]

3.2 PVFS list I/O

Da die klassische PVFS I Distribution lediglich zusammenhängenden Datenzugriff unterstützt, entwickelten Ching, Choudhary, Liao, Ross und Gropp die so genann-

te „list I/O“ Schnittstelle, die es ermöglicht nicht zusammenhängenden Datenzugriff durch einen einzigen Funktionsaufruf durchzuführen. Folgender Code-Ausschnitt stellt den Teil des Interfaces dar, der für das Lesen von Daten zuständig ist:

```
pvfs_read_list(   int mem_list_count,
                  char *mem_offsets[],
                  char mem_lengths[],
                  int file_list_count,
                  int file_offsets[],
```

Abbildung 8: PVFS list I/O Interface

Das Interface bietet einen Weg, Dateien sowie den Hauptspeicher unter Verwendung von Offsets in Regionen zu unterteilen. Diese Modellierung ermöglicht einen nicht zusammenhängenden Zugriff auf Speicher- und Dateiebene. Die Parameter des Funktionsaufrufes werden im Folgenden kurz erläutert [Ching 2002]:

- `mem_list_count` ist die Anzahl aller zusammenhängenden Speicherregionen, die in den Datenzugriff einbezogen werden sollen. Diese Anzahl legt auch die Länge der Arrays `mem_offset[]` und `mem_length[]` fest.
- `mem_offsets[]` ist eine Liste von Zeigern, die jeweils auf den Anfang einer zusammenhängenden Speicherregion zeigen.
- `mem_length[]` ist ein Array mit Längenangaben, die jedem Startpunkt einer zusammenhängenden Speicherregion eine gewisse Länge zuordnen.
- `file_list_count` hat die gleiche Funktion für eine Datei wie `mem_list_count` für den Speicher. Sie gibt die Anzahl der zusammenhängenden Regionen innerhalb einer Datei an und legt somit auch die Längen für `file_offset[]` und `file_length[]` fest.
- `file_offset[]` beschreibt ein Array von Offsets, von welchen jedes auf den Anfangspunkt einer zusammenhängenden Dateiregion zeigt.

`file_length[]` beinhaltet die Längen der Dateiregionen, die zu den Datei-Offsets gehören. Die Summe von `mem_length[]` und `file_length[]` muss gleich sein.

Fehler! Verweisquelle konnte nicht gefunden werden. stellt ein Beispiel für den Datentransfer über das list I/O Interface dar:

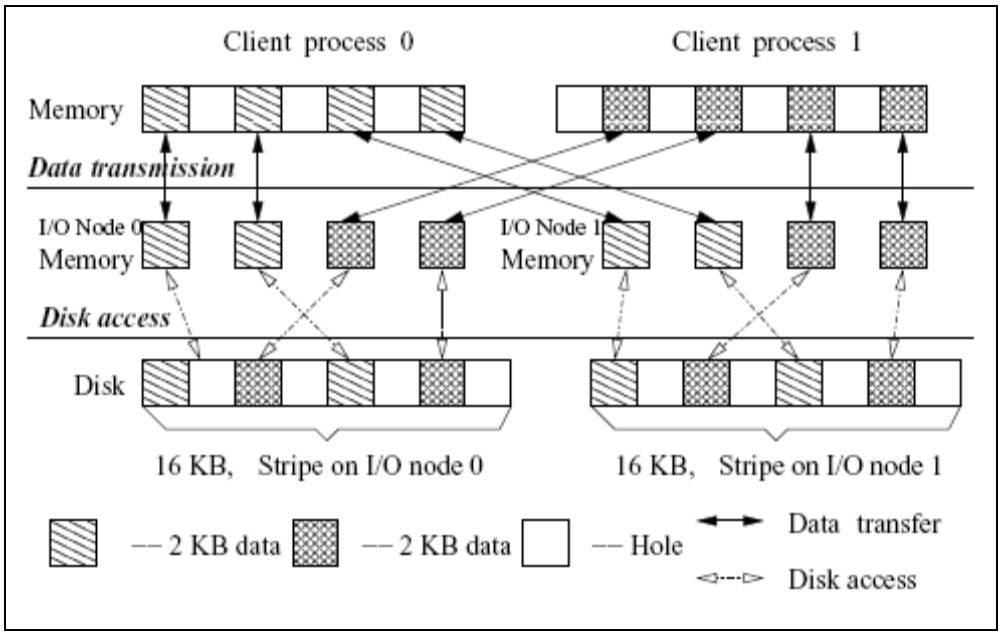

Abbildung 9: Datentransfer über PVFS list I/O [Wu 2003]

Viele konventionelle Kommunikationsprotokolle, wie zum Beispiel auch TCP/IP, unterstützen lediglich einen Datentransfer in zusammenhängenden Blöcken. Dabei können verschiedene Methoden verwendet werden, um Daten zwischen mehreren Puffern, die im list I/O Interface spezifiziert werden, zu transferieren.

Auf Grund der geringen Bandbreite traditioneller Kommunikationsschnittstellen werden Performance-Aspekte für nicht zusammenhängenden Datentransfer oft ignoriert. Ist das Netzwerk langsam, so fallen „Flaschenhälse“ wie zum Beispiel der Verbindungsaufbau nicht weiter ins Gewicht. Wird hingegen ein Netzwerk mit hoher Bandbreite und geringem Overhead verwendet, wie etwa *InfiniBand*, haben diese Aspekte einen hohen Einfluss auf die Gesamtperformance des Systems.

Es werden im folgenden Kapitel Möglichkeiten vorgestellt, wie effizienter nicht zusammenhängender Datentransfer in Hochgeschwindigkeitsnetzwerken erreicht werden kann.

3.3 Mechanismen für verteilten Dateitransfer

Wie im vorherigen Kapitel schon erwähnt, existieren verschiedene Methoden, um Daten zwischen Speicherpuffern auf Rechenknoten und I/O Knoten auszutauschen. Im Folgenden werden drei dieser Methoden beschrieben und gegenübergestellt. Alle dieser Methoden verwenden zur Datenübertragung RDMA Operationen.

3.3.1 Multiple Message

Die erste Methode für nicht zusammenhängenden Datentransfer wird als „Multiple Message“ bezeichnet. Hierbei wird für jeden zusammenhängenden Datenblock eine Nachricht versendet bzw. empfangen. In **Fehler! Verweisquelle konnte nicht gefunden werden.** wird dieser Vorgang verdeutlicht.

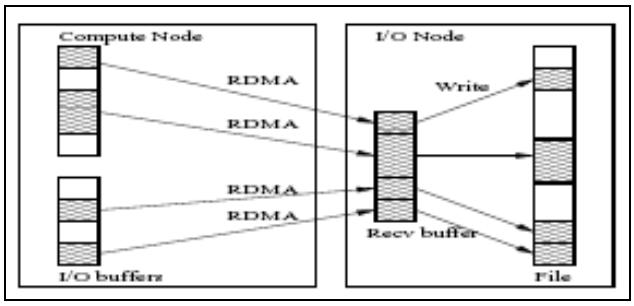

Abbildung 10: Multiple Message [Wu 2003]

3.3.2 Pack/Unpack

Ein zweites Schema zur Realisierung von nicht zusammenhängendem Transfer ist die „Pack/Unpack“ Methode. Es werden dabei die nicht zusammenhängenden Daten in einen temporären Puffer gepackt, dann übertragen und nach der Übertragung wieder entpackt. Abbildung 11 stellt dieses Verfahren dar.

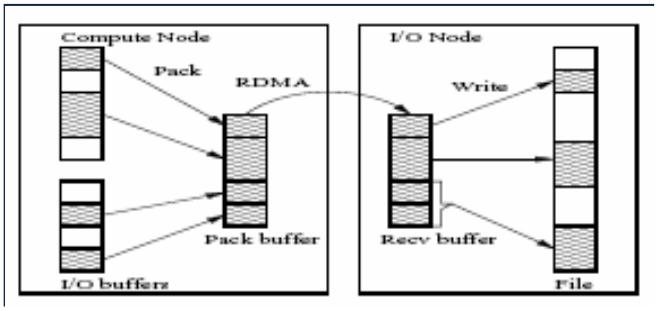

Abbildung 11: Pack/Unpack [Wu 2003]

3.3.3 Gather/Scatter

Für moderne Kommunikationsnetze wie InfiniBand, die RDMA Gather/Scatter Operationen unterstützen, kann ein weiteres Datentransferschema namens „Gather/Scatter“ verwendet werden. Durch nur einen Funktionsaufruf kann eine RDMA Schreiboperation mehrere Datensegmente sammeln und sie in einem Puf-

fer auf der Empfängerseite ablegen. RDMA Leseoperationen können Daten aus einem einzelnen Puffer auf der Gegenseite in mehrere Puffer auf der lokalen Instanz laden. In **Fehler! Verweisquelle konnte nicht gefunden werden.** wird eine RDMA Gather Schreiboperation verdeutlicht.

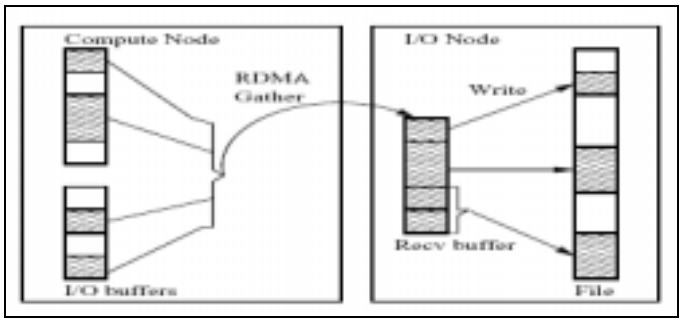

Abbildung 12: Gather/Scatter [Wu 2003]

Die Gather/Scatter Methode deckt die Anforderungen von PVFS list I/O sehr gut ab. Bei der Multiple Message Methode entstehen zum Zeitpunkt des Kommunikationsaufbaus hohe Performanceverluste, die durch die Gather/Scatter Methode drastisch reduziert werden können. Datenkopien, die beim Transfer mit Pack/Unpack hervorgerufen werden, können durch das Gather/Scatter Verfahren eingespart werden. [Wu 2003]

3.3.4 Probleme und Vergleich

Zwischen den drei beschriebenen Datentransfermethoden bestehen mehrere Kompromisse, die die Auswahl der geeignetesten Methode beim Systemdesign erschweren. Es werden hier einige dieser Kompromisse beschrieben.

Datenpuffer müssen bei Verwendung von InfiniBand registriert werden, bevor eine Datenübertragung stattfinden kann. Daher müssen bei den Methoden Multiple Message und Gather/Scatter alle list I/O Puffer registriert werden. Dahingegen muss bei Pack/Unpack lediglich der schon erwähnte temporäre Puffer registriert werden. In einigen Fällen ist es wünschenswert, diese Puffer nach vollendetem Datentransfer auch wieder freizugeben. Nun gilt es eine Entscheidung zu treffen, ob der Overhead von Speicherregistrierungen akzeptiert wird oder ob Datenkopien bei der Verwendung von Pack/Unpack vertretbar sind.

Die Anzahl der Datentransferoperationen variiert zwischen den verschiedenen Transfermethoden bei gleicher Datenmenge. Bei Multiple Message ist sie gleich der Anzahl der list I/O Puffer, wohingegen bei Pack/Unpack nur eine Operation durchgeführt werden muss. Bei Gather/Scatter hängt die Anzahl davon ab, wie viele Datensegmente in einer Transferoperation zusammengefasst werden können (momentan 64 bei InfiniBand). Jede Datentransferoperation ist mit einem Overhead beim Kommunikationsaufbau verbunden. Es muss daher entschieden werden, welche Methode für die entsprechende Anwendung am sinnvollsten ist.

Netzwerke, die RDMA verwenden, sind anfällig gegenüber der Pufferanordnung innerhalb des Speichers. Schlecht angeordnete Puffer können hohe Verzögerungen beim Datentransfer hervorrufen. Die Pack/Unpack Methode reserviert für ihren temporären Puffer eigenständig einen zusammenhängenden Pufferbereich. Werden die list I/O Puffer vom Benutzer definiert, kann nicht mehr gewährleistet werden, dass sie gemeinsam in einem zusammenhängenden Bereich angeordnet werden. Unter diesem Aspekt leidet die Performance von Multiple Message und Gather/Scatter.

Der hier durchgeführte Vergleich verdeutlicht, dass die Multiple Message Methode im Vergleich zu den anderen Methoden eine schlechtere Performance hat. Untersuchungen haben ergeben, dass das Packen von Daten und die Durchführung von Speicherregistrierungen die größten Auswirkungen auf die Performance haben. Auf Grund dieser Tatsache werden im nächsten Kapitel Vorgehensweisen zur effizienten Speicherregistrierung vorgestellt.

3.4 Effiziente Speicherregistrierung

Wie das vorangegangene Kapitel gezeigt hat, kann nicht zusammenhängender Datentransfer mit der Gather/Scatter Methode sehr effizient durchgeführt werden, wenn man den Aufwand für Speicherregistrierung und –deregistrierung reduziert. Es existiert hierfür eine Reihe von Möglichkeiten, die generell in die Klassen „Anwendungsgesteuerte Speicherregistrierung“ und „Bibliotheksgesteuerte Speicherregistrierung“ eingeordnet werden können. [Wu 2003]

3.4.1 Anwendungsgesteuerte Speicherregistrierung

Der erste Ansatz zur Optimierung der Speicherregistrierung ist die anwendungsgesteuerte Speicherregistrierung. Hier wird versucht den Vorgang der Speicherregistrierung in die Anwendungsebene zu verlagern. Es soll der Anwendung ermöglicht werden, selbst die Zuordnung des Speichers zu übernehmen.
Die PVFS Applikation erhält somit die Kontrolle über Speicherregistrierungsvorgänge und muss eigenständig die entsprechenden Bibliotheken aus dem PVFS Framework laden. Nachteile dieses Designansatzes sind, dass mehr Arbeitsaufwand in die Applikationsschicht verlagert wird und eventuelle Optimierungen durch die PVFS Bibliotheken nicht mehr möglich sind.
Eine weitere, nicht ganz so nachteilige Möglichkeit der anwendungsgesteuerten Speicherregistrierung ist es, die Kontrolle über den Speicher nur in einem gewissen Grad an die Anwendungsschicht zu übergeben. Dabei wird lediglich die zu reservierende Puffergöße an die PVFS Bibiothek übergeben und das Framework hat selbst die Möglichkeit die Zuteilung des Speichers zu optimieren. Das Problem der Anwendungsmodifikation bleibt dabei jedoch bestehen.

3.4.2 Optimistic Group Registration (OGR)

Neben der anwendungsgesteuerten Speicherregistrierung gibt es noch andere Mechanismen bei denen die PVFS Bibliothek selbst die Speicherregistrierung verwaltet und optimiert. In diesem Fall ist die PVFS Bibliothek völlig losgelöst von der Anwendungsebene, hat demnach allerdings auch keine Informationen darüber, wie der Anwendungsspeicher angeordnet ist. D.h. eine list I/O Operation könnte Speicherregionen verwenden, die weit außerhalb des Speicherbereiches der Anwendung liegen.

Ein Ansatz für die Optimierung von Speicherregistrierung und –deregistrierung trägt den Namen Optimistic Group Registration (OGR). OGR bietet die Möglichkeit list I/O Puffer zu sortieren und zu gruppieren. Es werden die Größen der Speicherregionen kontrolliert, die registriert werden können. Dadurch wird verhindert, dass große Lücken zwischen diesen Regionen entstehen. Es wird danach versucht jede dieser vordefinierten Speicherbereiche zu registrieren.
Dieses Schema hat den Vorteil, dass es transparent zur PVFS Applikation aufgebaut ist. Des Weiteren ist es im Allgemeinen sehr effizient, da sämtliche list I/O

Puffer in einem oder mehreren großen Speicherbereichen angelegt werden und wenig Lücken zu verzeichnen sind.

3.4.3 Vorgehensweise bei der Implementierung von OGR

Wie schon erwähnt werden die list I/O Puffer beim OGR Schema sortiert und gruppiert. Der Aufwand einer Pufferregistrierung wird mit folgendem Kostenmodell definiert [Wu 2003]:

$$T = a * p + b$$

Dabei ist a der Zeitaufwand einer Registrierung pro Seite, b der Overhead pro Operation und p die Größe der Puffer, angegeben in Seiten. Dieselbe Gleichung kann (mit anderen Werten für a und b) für die Deregistrierung verwendet werden. Anhand dieses Kostenmodells kann eine optimale Balance zwischen der Anzahl der Operationen und der zu verwendenden Puffergröße erreicht werden.

Der OGR Vorgang erfolgt in drei Schritten:

- Gruppierung der Puffer in Regionen, die auf dem Kostenmodell basieren und demnach Reduzierung von großen Speicherlücken
- Optimistische Registrierung jeder dieser ermittelten Regionen (d.h. es wird angenommen, dass der zu reservierende Bereich verfügbar ist)
- Falls das Betriebssystem die Registrierung verweigert, werden nicht zugewiesene Lücken herausgefiltert und es wird für die restliche Region mit individueller Registrierung fortgefahren (Bei den meisten Anwendungen ist dies jedoch selten der Fall).

Abbildung 13 verdeutlicht die Gruppierung von Puffern mit OGR. Puffer 1, 2 und 3 bilden in diesem Beispiel eine Gruppe. 5, 6 und 7 werden individuell registriert und der Bereich 4 wird herausgefiltert.

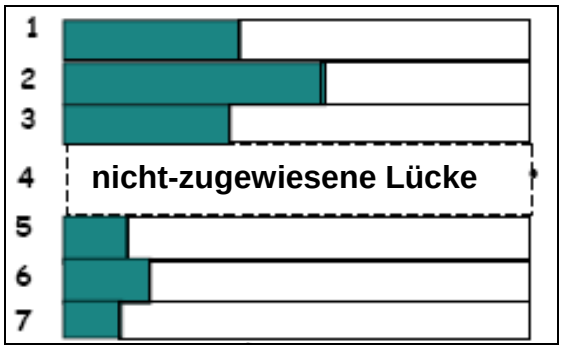

Abbildung 13: Gruppierung von Puffern mit OGR (in Anlehnung an [Wu 2003])

3.5 Ergebnisse und Auswertung

In der Literatur wird für die Analyse der Performance von Optimistic Group Registration ein Test vorgestellt, wobei ein 2-D Integerarray der Größe 2048x2048 zeilenweise (row-major-order) in eine Datei geschrieben wird. Das Array wird über Block-Scheduling auf vier Prozesse aufgeteilt. Jeder Prozess schreibt dabei seinen zugewiesenen Teil des Arrays in einen zusammenhängenden Bereich der Datei. Es treten bei diesem Vorgang keine Überlappungen innerhalb der Datei auf.

Für die Auswertung werden vier Fälle betrachtet:

- **ideal:** Hier wird keine Registrierung benötigt. Dies ist der Fall, wenn alle Pufferregistrierungen vorher im Cache abgelegt wurden.
- **individuell:** Hier werden alle Puffer individuell registriert.
- **OGR:** Es kommt hier das OGR Schema zum Registrieren von list I/O Puffern zur Anwendung.
- **OGR+Q:** Bei diesem Fall wird wiederum das OGR Schema angewendet. Es werden jedoch verschiedene nicht zugeordnete Lücken (vgl. Abbildung 13) simuliert, um Auswirkungen von Registrierungsfehlern hervorzuheben.

Tabelle 1 beinhaltet die Bandbreite der Schreibvorgänge, die Anzahl der Registrierungen und den Overhead der Registrierungen für jeden der vier Testfälle. Es wird des Weiteren unterschieden, ob der Schreibzugriff in Synchronisation mit dem Plattenzugriff erfolgt oder nicht.

Testfall	no sync (MB/s)	sync (MB/s)	# Reg.	Overhead (µs)
ideal	1010	82	0	0
individuell	424	73	1024	5254

OGR	950	≈82	1	227
OGR+Q	879	≈82	11	496

Tabelle 1: Benchmark OGR [Wu 2003]

Es wird deutlich, dass OGR im Vergleich zu individueller Registrierung den Aufwand der Registrierung von list I/O Puffern drastisch reduzieren kann. Gerade bei den Testläufen ohne Synchronisation werden große Unterschiede deutlich. [Wu 2003]

4 Zusammenfassung und Ausblick

Diese Seminararbeit stellt verschiedene Ansätze vor, die zur Performancesteigerung beim Einsatz des Clusterdateisystems PVFS I eingesetzt werden können.
Kapitel 2 umfasst hat drei Hauptaufgaben des Unifiers:

1) Unifier beseitigt überflüssige Daten im I/O-Pfad. Jeder Datengegenstand kann nur eine Kopie im ganzen System haben, die von allen Anwendungsbestandteilen und System-Subsysteme sicher und gleichzeitig genutzt wird. Ebenfalls beseitigt Unifier auch die vielfache Pufferung von Daten; dadurch wird die Größe des Caches effektiv vergrößert.
2) Unifier dient als Puffermanager, um Puffer für RDMA Operationen in den Netztechnologien zur Verfügung zu stellen. Es reduziert Speicherregistrierungs- und Deregistrationsaufwand und lässt dadurch alle Vorteile von RDMA Operationen zur Anwendung kommen.
3) Unifier stellt ein „Application-Level-Cache“ zur Verfügung, um anwendungsspezifische Optimierung zu erreichen. Durch die Schnittstellen wird eine bessere Zusammenarbeit zwischen den Bestandteilen unterstützt. [Wu 2003]

Experimentelle Ergebnisse von einem Prototyp zeigen Leistungsverbesserungen zwischen 30 und 70 % gegenüber der zwei anderen, in der PVFS I/O-Server-Implementierung häufig verwendeten Methoden.
Durch die PVFS I/O-Server wird eine bessere Skalierbarkeit erreicht. Infolge der Integration von Communication Buffer und den Cache Buffer steigert die Unifier-Methode auch die effektive Größe des Caches, was zu höherer Leistung führt.

Unifier wurde als Forschungsbestandteil im Design von PVFS2 eingesetzt. Die Integration von Unifier mit anderen PVFS2 Bestandteilen, sowie die Prüfung und Optimierung findet fortlaufend statt. Die potenziellen Vorteile werden erweitert.

Kapitel 3 gab einen Überblick über die Problemstellung nicht zusammenhängenden Datenzugriffs. Es wurde der Ansatz Optimistic Group Registration zur Optimierung von Pufferregistrierungsvorgängen vorgestellt. Durch einen Benchmark-Test konnte gezeigt werde, dass durch Optimistic Group Registration der Aufwand der Speicherregistrierung bei nicht zusammenhängendem Datenzugriff stark reduziert werden kann.

Literaturverzeichnis & weiterführende Literatur

[Ching 2002] Ching, A.; Choudhary, A.; Liao, W.; Ross, R.; Gropp, W.: Non-contiguous I/O through PVFS, Evanston, 2002

[InfiniBand 2005] InfiniBand® Trade Association Specifications FAQ, Portland 2005.
http://www.infinibandta.org/specs/faq/ (16.06.2005)

[VFS 2002] Verteilte File System -ein Wissenschaftliches Projekt des Leibniz-Rechenzentrums der Bayerischen Akademie der Wissenschaften, München 2002.

[Wu 2003] Wu, J.; Wyckoff, P.; Panda, D.: Supporting Efficient Noncontiguous I/O through PVFS over InfiniBand, Columbus, 2003

[Wu 2003] Jiesheng Wu, Pete Wyckoff, Dhabaleswar Panda, Rob Ross; Unifier: Unifying Cache Management and Communication Buffer Management for PVFS over InfiniBand, Columbus 2003.

[Wikimedia 2005] Jimmy Wales, Wikimedia Foundation Inc., USA Juni 2005.
http://de.wikipedia.org/wiki/InfiniBand (14.06.2005)

[PVFS 2005] PVFS Homepage: The Parallel Virtual File System Project, Clemson University 2005.
http://www.parl.clemson.edu/pvfs/index.html (13.06.2005)

[ZeroCopy 2005] ZeroCopy TCP/IP Protocol Stack, Cyclone Microsystems, New Haven 2005.
http://www.cyclone.com/products/zerocopy.php (17.06.05)