Fachhochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich Informatik
Department of Computer Science

Seminararbeit

im Master Studiengang
Veranstaltung: Parallele Systeme

Parallel Virtual File System (PVFS) I

- Architektur, Verwendung und Eigenschaften -

von
Björn Adam
Christian Staron

Seminarleiter: Prof. Dr. R. Berrendorf

Eingereicht am: 22. Mai 2005

Inhaltsverzeichnis

Inhaltsverzeichnis II
Abbildungsverzeichnis III
Abkürzungsverzeichnis IV
1 Verteilte Dateisysteme 1
1.1 Klassischer Ansatz von verteilten Dateisystemen 1
1.2 Paralleler Ansatz von verteilten Dateisystemen 2
2 PVFS – Parallel Virtual File System 3
2.1 Einleitung 3
2.2 Aufbau 3
2.3 Komponenten 4
2.4 Eigenschaften & Verwendung von PVFS I 6
2.4.1 Native PVFS – API 8
2.4.2 Linux Kernel Interface (Virtual File System (VFS) Module) 8
2.4.3 MPI-IO & MPI-2 8
2.5 Installation, Konfiguration & Einsatz eines PVFS I Dateisystems 9
2.5.1 Installation des Metadatenservers 10
2.5.2 Installation der I/O – Knoten 10
2.5.3 Installation der Rechenknoten 10
2.5.4 Systemstart und Shutdown 11
2.6 Installation eines Testsystems & Performanceergebnisse 11
2.6.1 Simultanes Lesen und Schreiben mit native PVFS auf Fast Ethernet und Myrinet 12
2.6.2 Simultanes Lesen und Schreiben mit native PVFS und MPI-IO 15
2.6.3 Performanceanalysen 16
3 Fazit zu PVFS – Version I und Ausblick zu Folgeentwicklungen 17
Literaturverzeichnis V

Abbildungsverzeichnis

Abbildung 1: Struktur eines NFS 1
Abbildung 2: Struktur eines verteilten Dateisystems 2
Abbildung 3: PVFS System Aufbau 4
Abbildung 4: Zugriff auf Metadaten 5
Abbildung 5: Zugriff auf physikalische Daten 5
Abbildung 6: Datenzugriff über PVFS Linux Kernel Support 6
Abbildung 7: PVFS Lesegeschwindigkeit – Fast Ethernet (4 bis 32 I/O Server) 13
Abbildung 8: PVFS Schreibgeschwindigkeit – Fast Ethernet (4 bis 32 I/O Server) 13
Abbildung 9: PVFS Lesegeschwindigkeit – Myrinet (4 bis 32 I/O Server) 14
Abbildung 10: PVFS Schreibgeschwindigkeit – Myrinet (4 bis 32 I/O Server) 15
Abbildung 11: PVFS Performance – native PVFS vs. ROMIO MPI-IO Implementierung 16

Abkürzungsverzeichnis

API	Application Programming Interface
ca.	circa
Daemon	Disk and execution monitor
GByte	Gigabyte
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
TCP	Transmission Control Protocol
US	United States
VFS	Virtual File System

1 Verteilte Dateisysteme

Die immer stärker werdende Verwendung von PC Clustern in der Forschung und der Wirtschaft führt dazu, dass die Anforderungen an diese Systeme ebenfalls steigen. Die Prozessorhersteller entwickeln immer schnellere Chips, um dem Trend folgen zu können. Das Performance Problem liegt allerdings weniger in der Leistungsfähigkeit der Prozessoren als in dem des Transfers von Daten. Es entstehen besonders im Fall von großen Datenmengen Engpässe (im Folgenden: *Bottlenecks)* im Bereich der Eingabe und Ausgabe von Daten, da die Daten in diesem Fall auf das Dateisystem ausgelagert werden müssen.
Ein populärer Ansatz diese Effekte zu mildern sind die Parallelen Datei Systeme. Ein Beispiel für ein paralleles Dateisystem ist das *Parallel Virtual File System (PVFS),* welches in der folgenden Seminararbeit kurz vorgestellt und erläutert werden soll.

1.1 Klassischer Ansatz von verteilten Dateisystemen

Der klassische Ansatz von globalen Dateisystemen ist der Client/Server Ansatz. Ein Beispiel dafür ist das NFS (Network File System).

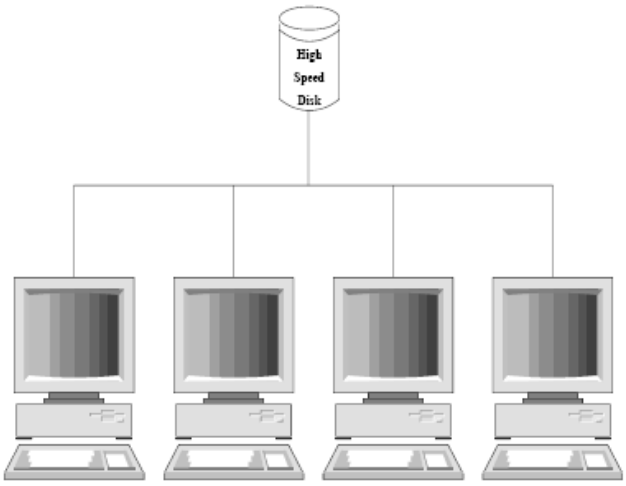

Abbildung 1: Struktur eines NFS

[Quelle: Schütt 2001]

Die Abbildung zeigt, dass die Clients auf einen zentralen Server zugreifen. Daraus resultieren mehrere Nachteile. Das größte Problem ist das Auftreten von Bottlenecks an dem Anschluss des Servers an das Netzwerk. Die Bandbreite der Netzwerkkarte am Server ist be-

grenzt, zusätzlich müssen sich alle Clients diese Bandbreite teilen. Die Lastverteilung ist dadurch sehr schlecht.

1.2 Paralleler Ansatz von verteilten Dateisystemen

Ein verbesserter Ansatz ergibt sich aus der Festlegung die Daten nicht zentral auf einem Server zu lagern, sondern dezentral über den gesamten Cluster zu verteilen.

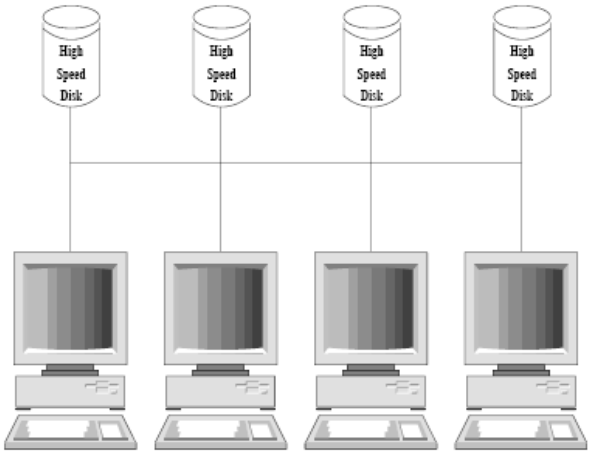

Abbildung 2: Struktur eines verteilten Dateisystems

[Quelle: Schütt 2001]

Man erkennt sehr schnell, dass die Lastverteilung durch die verteilte Datenablage deutlich ausgeglichener ist.

Generell ergeben sich drei Hauptanforderungen an ein paralleles Dateisystem die in der Praxis mittels drei verschiedenen Diensten erfüllt werden könnten:

- Dienst, der für einen konsistenten Namensraum sorgt,
- Dienst, der die Verteilung von Daten über den Cluster ausführt,
- Dienst, der zusätzliche Eingabe/Ausgabe Interfaces liefert.

Der konsistente Namensraum soll dafür sorgen, dass Daten auf mehreren Knoten des Clusters zugreifbar sind. Die zusätzliche physikalische Verteilung der Daten führt zu einer Lastverteilung im Vergleich zu einem Client/Server Ansatz. Die zusätzlichen Eingabe/Ausgabe Interfaces ermöglichen den Benutzern weiterführende Funktionen wie z.B. neue Zugriffsmodi.

Im Folgenden wird auf ein Beispiel für ein paralleles Dateisystem näher eingegangen: das PVFS.

2 PVFS – Parallel Virtual File System

2.1 Einleitung

Das PVFS ist ein Ansatz ein skalierbares paralleles Dateisystem auf Unix Systemen zu realisieren. Da PFVS eine User-Level Implementierung ist, sind zum Betreiben des Systems keine Änderungen am Linux-Kernel notwendig. PVFS steht unter der GNU-Lizenz und ist somit frei verfügbar.

Parallel: Daten werden verteilt auf den einzelnen Knoten des Clusters abgespeichert.

Virtual: Das Dateisystem stellt nur ein virtuelles Dateisystem zur Verfügung. Die Daten werden über eine auf dem Cluster laufende Programminstanz auf das lokale Dateisystem gespeichert.

File **S**ystem: Das Dateisystem stellt dem Benutzer die üblichen Funktionen (öffnen, lesen, schreiben, schließen) zur Verfügung.

2.2 Aufbau

Der PVFS-Ansatz erfüllt die Anforderungen an ein verteiltes paralleles Dateisystem.
Es besitzt folgende grundlegenden Eigenschaften:

- einen konsistenten Namensraum
- einen für Anwendungen transparenten Zugriff auf die Daten
- Eine physikalische Verteilung der Daten im PC Cluster

Damit das parallele Dateisystem effizient genutzt werden kann, muss es einen homogenen Namensraum zur Verfügung stellen, der über das PC-Cluster hinweg gleich ist. Allen Knoten im PC Cluster muss dieselbe Verzeichnis Struktur mit denselben Dateninhalten verfügbar gemacht werden. [PVFS 2005]

Die Daten werden bei PVFS physikalisch redundant abgespeichert. Da es dadurch mehrere Pfade gibt, die Daten zu erreichen, werden die Performance und die Lastverteilung verbessert. Die kumulierte Bandbreite wird erhöht und einzelne Bottlenecks eliminiert. [PVFS 2005]

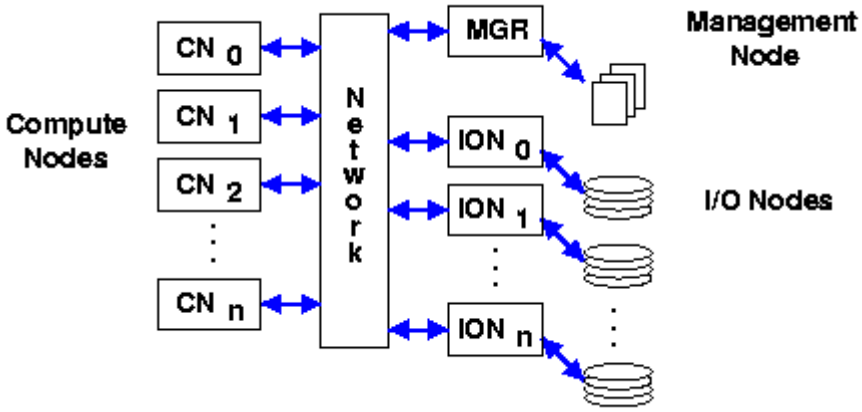

Abbildung 3: PVFS System Aufbau

[Quelle: PVFS 2005]

Jedem Knoten im PC Cluster ist eine spezielle Aufgabe zugeordnet. Ein Knoten kann entweder ein Rechenknoten sein auf dem die Anwendungen laufen, ein Managementknoten sein, der für die Metadaten-Operationen zuständig ist oder ein I/O Knoten sein der dafür zuständig ist die Daten physikalisch zu speichern. Die obige Abbildung illustriert diese Aufteilung. Die Management- und I/O – Knoten können ebenfalls für Berechnungen genutzt werden. Bei kleinen PC Clustern kann es sinn machen die einzelnen Aufgaben auf einem Knoten laufen zulassen, in einem großen PC Cluster empfiehlt es sich die einzelnen Dienste dediziert zu betreiben. [PVFS 2005]
Im Folgenden wird auf die zentralen Komponenten des PVFS Systems kurz eingegangen.

2.3 Komponenten

Das PVFS besteht aus vier Haupt Komponenten:

- Metadaten Server Knoten (mgr)
- I/O Server Knoten (iod)
- PVFS native API (libpvfs)
- PVFS Linux kernel support

Diese einzelnen Komponenten können wie schon erwähnt als Dienst auf einem Knoten im Cluster betrieben werden. [PVFS 2005]

Die ersten beiden Komponenten laufen als Dienste auf den Knoten des PC Clusters.
Der Metadaten Server organisiert die Metadaten der Dateien, die auf dem PVFS abgelegt sind. Die Metadaten beschreiben die Eigenschaften der Daten, wie z.B. Dateiname, Ablageort in der Verzeichnisstruktur, den Besitzer und die Art wie die Datei physikalisch über

das PC-Cluster verteilt ist. Dadurch, dass ein Dienst die Verwaltung der Metadaten übernimmt, bleiben die Metadaten auch bei multiplem Zugriff konsistent und es bedarf keinen komplizierten Locking Mechanismen wie dies bei einem klassischen Netzwerk Dateisystem üblich ist. [PVFS 2005]

Die zweite Komponente ist der I/O Server. Der I/O Server Dienst ist für das physikalische Speichern und Laden der Daten verantwortlich. Knoten, die den I/O Server Dienst ausführen, speichern Daten physikalisch auf die Ihre lokalen Festplatten. Der I/O Server dient als „Zwischenschicht" oder als Verbindungsglied zwischen dem PVFS und dem lokalen Dateisystem. Daraus folgt das, dass lokale Dateisystem weitestgehend unabhängig von dem PVFS ist. Dadurch hat man die Möglichkeit völlig unabhängig, lokale Sicherheitsmechanismen wie z.B. RAID Systeme auf den Knoten einzusetzen, um die Fehlertoleranz zu minimieren. [PVFS 2005]

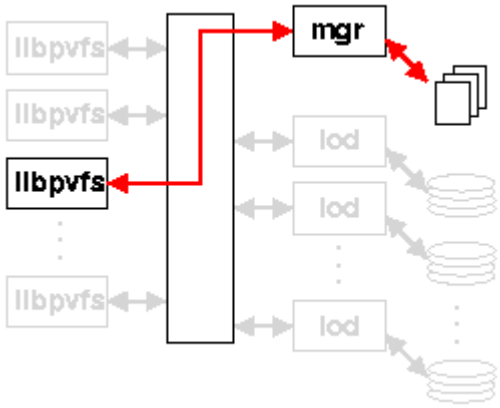

Abbildung 4: Zugriff auf Metadaten

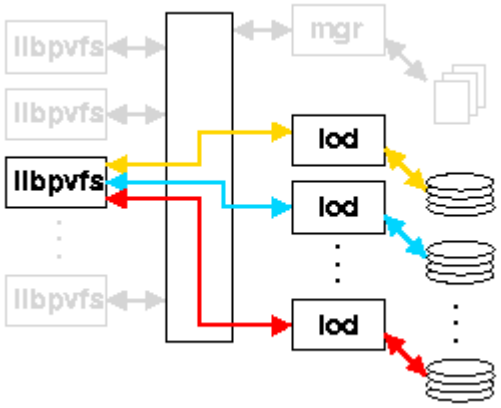

Abbildung 5: Zugriff auf physikalische Daten

[Quelle: PVFS 2005]

Die dritte Komponente ist die PVFS native API (libpvfs). Diese API stellt die Verbindung zwischen den Metadaten und den physikalischen Daten her. Die obigen Abbildungen erläutern das vorliegende Prinzip.

Bei Operationen die, die Metadaten betreffen, kommunizieren die Anwendungen über die libpvfs mit den Metadaten Servern. Bei einem reinen Datenzugriff wird der Metadaten Server nicht benötigt. Die einzelnen I/O Server kommunizieren, über die libpvfs, direkt miteinander. Dieser Ansatz ermöglicht eine gute Skalierbarkeit des gesamt Systems. [PVFS 2005]

Die letzte Komponente ist der PVFS Linux kernel support. Dieser Dienst ist notwendig um das PVFS-Dateisystem auf Linux Knoten zu mounten. Die Komponente ermöglicht den auf dem Linux Knoten laufenden Anwendungen einen transparenten Zugriff auf das PVFS Dateisystem. Der transparente Zugriff hat den großen Vorteil, dass die Anwendungen in keiner Weise geändert werden müssen, um mit dem PVFS-Dateisystem arbeiten zu können. [PVFS 2005]
Die untere Abbildung illustriert die Methode des Datenzugriffs unter Verwendung von PVFS. Ein Dienst mit dem Namen pvfsd (Parallel Virtual File System – Daemon) übernimmt im Namen der Anwendungen den Zugriff auf das PVFS-Dateisystem. Der Dienst benutzt dazu Funktionen aus der PVFS native API. [PVFS 2005]

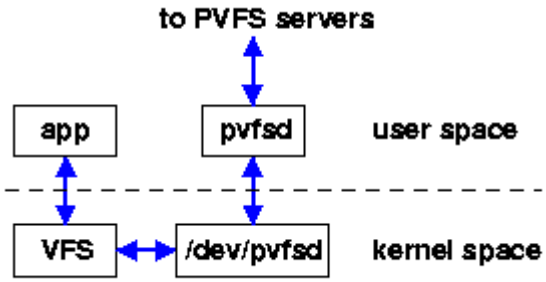

Abbildung 6: Datenzugriff über PVFS Linux Kernel Support

[Quelle: PVFS 2005]

Die Anfragen der Anwendungen werden durch die Linux VFS (Virtual File System) – Schicht durchgeschleust um anschließend von dem pvfsd Dienst angenommen zu werden. Ausstehende Anfragen werden in einer Schlange verwaltet („gequeued"). Der pvfsd Dienst fordert dann nach und nach die gewünschten Daten von den PVFS-Servern an und liefert die Daten an die Anwendungen. [PVFS 2005]

2.4 Eigenschaften & Verwendung von PVFS I

Wie bereits in Kapitel 2.2 beschrieben, erfüllt PVFS I allgemeine Anforderungen eines verteilten Dateisystems. Betrachtungspunkte sind dabei die Gewährleistung eines im Netzwerk konsistenten Namensraums, den für Benutzer transparenten Dateizugriff und die physikalische Verteilung der zu speichernden Daten innerhalb des betrachteten Clusters.

Insbesondere bezüglich des letzten Punktes, der Verteilung der Daten, lässt sich erklären, dass bei der Entwicklung von PVFS I die Performance des Dateisystems eine wesentliche Rolle spielte, sodass die Entwickler das Modell auf Basis eines parallelen Disk Schemata aufbauten. Nach diesem Konzept können eine Reihe unabhängiger Datenträger verwendet werden, um einen simultanen Zugriff auf diese zu Unterstützen und somit die Bandbreiten der Einzelgeräte bei Messungen aggregiert werden können. Auf diese Weise kann der Bedarf seitens Industrie und Wissenschaft nach effizienten Speichersystemen für große Datenvolumina gedeckt werden, was im Kapitel 2.6 verdeutlicht wird. [Rajaram 2002, S. 8f] Darüber hinaus werden bei der Verteilung der Daten auf dem Cluster weitere, durch PVFS I implementierte Mechanismen angesprochen und verwendet. Bei der Verteilung werden die Daten in Blöcke aufgeteilt und diese über das Netzwerk verteilt auf den Einzelgeräten, ähnlich einem RAID („Redundant Array of Independent Disks“) – System der Ebene 0, gespeichert. Dabei hat der Benutzer die Möglichkeit die Blockgröße, die typischerweise 64 kByte beträgt, zu bestimmen. [Carns 2002; Ligon III 1999, S. 2]

Um die im vorigen Abschnitt spezifizierten physikalischen Dateiparameter und weitere Eigenschaften der gespeicherten Daten prüfen zu können, werden seitens PVFS Dienstprogramme zur Verfügung gestellt. Anhand dieser sind Informationen zu Streifen- beziehungsweise Blockgrößen der verteilten Dateien, der Dateiverteilung im Allgemeinen, Serverstatus, beispielsweise zu den I/O Servern, sowie Informationen zu den PVFS – Metadaten des Managementservers abrufbar und zum Teil konfigurierbar. [Carns 2002]

Jegliche Kommunikation der Daten innerhalb des PVFS I – Systems wurde anhand des Transmission Control Protocol (TCP) realisiert. Dabei werden seitens kommunizierender Anwendungen TCP – Verbindungen zu den PVFS – I/O Knoten aufgebaut. Die Lebenszeit einer Verbindung ist dabei abhängig von der Lebenszeit der initiierenden Applikation, beziehungsweise von den Zeitgrenzen für offene TCP – Verbindungen. [Carns 2000; Ligon III 1999, S. 2f]

Außerdem weist ein PVFS I – System bemerkenswert flexible Möglichkeiten zur Verwendung auf. Prinzipiell lassen sich die Zugriffsvarianten in drei Ansätze aufteilen, die in folgenden Abschnitten detaillierter erläutert und voneinander abgegrenzt werden. [Argonne 2005; Carns 2000; Carns 2002; Ligon III 1999, S. 2ff; Rajaram 2002, S. 1f, 9-19, 21-32]

- Native PVFS – API
- Linux Kernel Interface
- MPI-IO & MPI2-IO – Programmierschnittstelle

2.4.1 Native PVFS – API

Ein besonderer Vorteil zum Einsatz von PVFS ist die Flexibilität in der Verwendung, sodass bereits existierende compilierte Programme ohne jegliche Veränderung auf dem verteilten Dateisystem arbeiten können. Sofern diese Applikationen die Standard UNIX I/O Funktionen (zum Beispiel: „read()", „write()") oder UNIX Shell Kommandos programmintern verwenden, werden entsprechende Systemaufrufe durch einen „Trapping" – Mechanismus abgefangen und durch eine PVFS konforme C – Bibliothek mit „Wrapper" Funktionen adaptiert. Diese PVFS Client – Bibliothek muss dazu lediglich vor Ausführung der Anwendung durch eine Umgebungsvariable, wie zum Beispiel „LD_PRELOAD", dynamisch gelinkt werden, um den „Falltür – Mechanismus" zu ermöglichen. Nachteile dieses Verfahrens werden in zwei Beispielen sichtbar. Einerseits besteht die Gefahr, durch das Abfangen eines Systemaufrufs, wie beispielsweise „exec()" den aktuellen Benutzerstatus im Datenspeicher der Applikation zu zerstören, wodurch der neue Prozess nicht in der Lage ist, eine logische Beziehungen zu zuvor geöffnete PVFS Dateien im Datenspeicher aufrechtzuerhalten. Die zweite Schwierigkeit des Verfahrens erklärt sich in der Portierbarkeit dieses Ansatzes auf andere Architekturen und Betriebssysteme. Dabei müssen die benötigten Systemaufrufe identifiziert und in der PVFS Client – Bibliothek entsprechend in Wrapper Funktionen umgesetzt werden. Darüber hinaus ist dieser Anpassungsprozess ebenso nötig, wenn sich Programmierschnittstellen von Systembibliotheken ändern. [Argonne 2005; Carns 2000; Carns 2002; Ligon III 1999, S. 2, 4]

2.4.2 Linux Kernel Interface (Virtual File System (VFS) Module)

Diese Alternative wird durch Erweiterungsmöglichkeiten des Linux Kernels realisiert. Der Betriebssystemkern unterstützt durch das individuelle Laden von Modulen eine Einbindung von alternativen Dateisystemen, ohne den Kern dafür compilieren zu müssen. Auf diese Weise kann ein PVFS Dateisystem in das Betriebssystem „eingehangen" und für gewöhnliche Dateizugriffe verwendet werden. [Argonne 2005; Carns 2000]

2.4.3 MPI-IO & MPI-2

Die MPI (Message Passing Interface) Programmierschnittstelle gilt heute als „de facto" Standard zur Programmierung von Multicomputer- und Multiprozessorarchitekturen, wobei diese verteilt in Form eines Clusters oder integriert in einer Rechnerarchitektur vorliegen können. MPI-IO ist dabei aufsetzend eine Erweiterung des MPI – Standards um eine Reihe von parallelen I/O Operationen und Optimierungsansätzen. Beispielfunktionalitäten sind dabei nicht benachbarte Dateizugriffe, kollektives I/O, asynchrones I/O, geteilte Da-

teizeiger und portable Datenrepräsentation. Außerdem erlauben diese Operationen weit reichende Optimierungsansätze, da sich dadurch überlappende I/O Zugriffe und Kommunikationen implementieren lassen. Beispielszenarios erlauben demnach die Ein-/Ausgabe unabhängig von Berechnungsaktivitäten anderer Rechenknoten und die damit ermöglichte maximale Ressourcennutzung, sodass die Ausführungszeit des Prozesses durch die effiziente Auslastung aller Hardware- und Softwarekomponenten minimiert werden kann. 1997 wurden die Operationen in den MPI-2 Standard aufgenommen, sodass sie bei heutigen MPI basierenden Implementierungen verwendet werden können. Demnach zeichnet sich MPI-IO als portable Programmierschnittstelle für parallele I/O Lösungen aus, die auf einer Vielzahl von kommerziellen und aus der Forschung entstandenen Dateisystemen einsetzbar ist. Weiterführend dominieren zwei wesentliche Ansätze zur Verwendung von Dateisystemen durch MPI-IO: der portable Ansatz und der Punkt zu Punkt Ansatz. Ein typischer Vertreter des Ersten ist ROMIO, welcher die I/O Implementierung durch eine dem Dateisystem aufsetzende Schnittstelle mit einem dementsprechenden Overhead zugunsten der Portabilität realisiert. Ein Beispiel für den zweiten Ansatz ist GPFS, wodurch die Implementierung unmittelbar an das darunter liegende Dateisystem geknüpft wird und somit durch weniger Overhead realisiert werden kann. Insbesondere die ROMIO Implementierung führte durch ihre leichte Portierbarkeit auf neue Dateisysteme zum effizienten Einsatz von PVFS und stellt damit eine häufig verwendete Programmierschnittstelle zur Operation auf dem Dateisystem dar. [Argonne 2005; Carns 2000; Ligon III 1999, S. 3; Rajaram 2002, S. 1f, 9-19, 21-32]

2.5 Installation, Konfiguration & Einsatz eines PVFS I Dateisystems

Generell beschreibt sich das PVFS I Dateisystem als eine „Benutzerlevel – Implementation", sodass für die Installation und den Betrieb von PVFS keine Betriebssystemkern – oder Modul – Änderungen nötig sind. Jedoch handelt es sich um ein Multikomponentensystem, was eine gewisse Grundkomplexität verspricht und demnach seitens des Administrators für die Installation und die Konfiguration Sachverstand und Geduld erfordert. Gemäß der Beschreibung in Kapitel 2.3 existieren verschiedene Komponenten mit unterschiedlichen Rollen für die Aufgaben im Gesamtsystem. Dabei können alle Rollen auf einer Architektur gemeinsam oder über verschiedene Knoten verteilt installiert werden. Mit Installation der Software auf einem Clusterknoten werden seine Funktionalitäten im System fixiert. Dabei ist zu beachten, dass die Rolle des Metadatenservers (auch: „Head – Knoten") nur einmal vergeben wird, wogegen es viele I/O Server und viele Clients, beziehungsweise Rechenknoten innerhalb eines Systems geben kann. In folgenden Abschnitten wird die In-

stallation und Konfiguration der verschiedenen Systemkomponenten, sowie deren Funktionalität erläutert. [Carns 2000; Carns 2002; Ligon III 1999, S. 2]

2.5.1 Installation des Metadatenservers

Wie einleitend beschrieben, spielt der Metadatenserver eine zentrale Rolle im Gesamtsystem, sodass er auf lediglich einem Knoten realisiert wird. In einem Verzeichnis speichert er Informationen zu den über das PVFS Dateisystem gespeicherten Dateien. Diese umfassen beispielsweise Angaben zu den Dateieignern, Rechten, und wie die Dateien über die I/O Knoten verteilt sind. Darüber hinaus werden zwei Dateien beim Metadatenserver gespeichert, die benötigt werden, um I/O Server und im PVFS abgespeicherte Dateien wieder zu finden. Aus diesen Angaben lassen sich die Funktionen des Head Knotens ableiten. Er ist verantwortlich für die Verwaltung der gespeicherten Daten, enthält Informationen zu deren Verteilung innerhalb des PVFS, regelt die Zugriffsrecht auf die Daten und verwaltet die Strukturinformationen zu den I/O Servern des gesamten PVFS. [Carns 2002]

2.5.2 Installation der I/O – Knoten

Beim Aufsetzen eines I/O Knotens wird ein Daemon installiert, der den I/O Service im Cluster bereitstellt. Für den Betrieb des Servers wird außerdem mindestens ein Datenverzeichnis eingerichtet, welches dazu verwendet wird, als PVFS Dateien markierte Daten zu speichern. In einer Konfigurationsdatei werden die auf der Maschine befindlichen Datenverzeichnisse verwaltet. [Carns 2002]

2.5.3 Installation der Rechenknoten

Zentrale Rolle der Rechenknoten ist, auf ihnen ausgeführte Applikationen die PVFS – Funktionalitäten bereitzustellen. Dementsprechend werden auf ihnen Daemons installiert, die für anfordernde Anwendungen den Netztransfer innerhalb des PVFS Systems zur Verfügung stellen und realisieren. Um die Funktionen für den Anwender beziehungsweise die Anwendung transparent zu halten, muss auf einem solchen Knoten der Daemon lauffähig installiert und gestartet sein, bevor das PVFS Dateisystem seitens des Clients „gemountet" werden kann. Entsprechend der Verwendungsmöglichkeiten aus Kapitel 2.4 müssen für etwaige Programme die Zugriffsmöglichkeiten auf das PVFS System in Form eines Kernelmoduls, einer PVFS Client Bibliothek oder die MPI-IO Schnittstelle, installiert sein. Wurde das PVFS Dateisystem in einem Verzeichnis „eingehangen", kann, wie bei gewöhnlichen Dateisystemen darauf zugegriffen und gearbeitet werden. [Carns 2002]

2.5.4 Systemstart und Shutdown

Um das installierte PVFS System zu verwenden, müssen alle Server gestartet werden. Die Reihenfolge, in der sie gestartet werden, ist dabei unerheblich. Danach kann die Clientsoftware initialisiert werden und das PVFS System in die Clientverzeichnisse „gemountet" werden, was das System einsatzbereit macht. [Argonne 2005; Carns 2002]
Greifen keine Rechenknoten mehr auf das Dateisystem zu, kann jeder das PVFS System aushängen, sodass die Server wieder heruntergefahren werden können. [Argonne 2005; Carns 2002]

2.6 Installation eines Testsystems & Performanceergebnisse

Um die Performance eines PVFS Dateisystems aufzuzeigen, wird folgend eine Testinstallation auf dem Chiba City Cluster des Argonne National Laboratory, Illinois USA dargestellt und die Resultate der Performanceanalysen beschrieben.
Der Cluster besteht aus 256 Knoten mit jeweils zwei 500 MHz Pentium III Prozessoren, 512 MByte Hauptspeicher und 9 GByte Quantum Atlas IV SCSI Festplatten. Die Festplatten wiesen bei Prüfungen durch ein „Bonnie" Dateisystem – Benchmark eine Schreibgeschwindigkeit von 22 MByte pro Sekunde und eine Lesegeschwindigkeit von 15 MByte pro Sekunde, bei sequenziellem Zugriff auf eine 512 MByte große Datei, auf. Verbunden wurden die Knoten durch einerseits 100 Mbit pro Sekunde Intel EtherExpress Pro Fast Ethernet Netzwerkkarten im vollen Duplex – Modus und andererseits 64 Bit Myrinet (Revision 3) – Karten. Auf den Knoten arbeitete ein Linux Betriebssystemkern in der Version 2.2.15pre4. Für die verschiedenen Netzwerke wurden unterschiedliche MPI Implementierungen verwendet: MPICH 1.2.0 für das Fast Ethernet Netzwerk und MPICH-GM 1.1.2 für das Myrinet Netz. Die Linuxkernel wurden für Singleprozessoren compiliert, sodass für die Tests nur eine CPU auf einem Knoten arbeitete. Des Weiteren wurden für die Experimente lediglich 60 Knoten der 256 verwendet, von denen einige als Rechenknoten und I/O Server für das PVFS System verwendet wurden. Da TCP für die Kommunikation innerhalb des Dateisystems verwendet wurde, wurde die TCP Bandbreite vor Beginn der Experimente an beiden Clusternetzen gemessen. Dabei wurde die TCP Testversion ttcp 1.1 genutzt. Darüber hinaus wurden drei TCP – Buffergrößen mit 8 KByte, 64KByte und 256 KByte ausprobiert. Für alle drei Größen konnten jedoch ähnliche Ergebnisbandbreiten von 10,2 MByte pro Sekunde bei dem Fast Ethernet und 37,7 MByte bei dem Myrinet ermittelt werden. Für die PVFS Performancemessungen wurden Tests aus folgenden Bereichen entwickelt:

- Simultanes Lesen und Schreiben auf Basis der native PVFS – API und dem Fast Ethernet Netzwerk
- Simultanes Lesen und Schreiben auf Basis der native PVFS – API und dem Myrinet Netzwerk
- Simultanes Lesen und Schreiben mit der native PVFS – API und der MPI-IO Programmierschnittstelle

Für alle Messungen wurde die PVFS – Dateiblockgröße von 16 KByte verwendet. Die Experimente basieren auf einem Testprogramm, welches für die Tests aus 2.6.1 durch eine native PVFS – API implementiert ist und für die Messungen aus 2.6.2 MPI-IO Funktionsaufrufe verwendet. Das Testprogramm öffnet dabei jeweils eine neue PVFS – Datei, schreibt in diese simultan mit allen Rechenknoten in disjunkte Regionen Daten, schließt die Datei, öffnet sie wieder und liest simultan alle Daten aus der Datei und schließt sie danach wieder. Die Zeitmessungen finden dabei jeweils an jedem Rechenknoten statt, um sie danach aggregieren zu können. Das zu schreibende und zu lesende Datenvolumen variiert bei den Tests in Abhängigkeit von der in dem jeweiligen Testfall eingestellten Anzahl der I/O Server. Dabei muss jeder Prozess ein Datenvolumen pro Operation von 2 * „Anzahl I/O Server" MByte Daten verarbeiten. Jeder Test wurde fünf Mal hintereinander durchgeführt. Die größte und die kleinste Messung wurden dabei jeweils verworfen. Aus den verbleibenden drei Messresultaten wurden durch Bildung der arithmetischen Mittel die endgültigen Resultate berechnet. [Carns 2000]

2.6.1 Simultanes Lesen und Schreiben mit native PVFS auf Fast Ethernet und Myrinet

Fast Ethernet

Die folgenden Abbildungen zeigen das Leistungsverhalten beim Schreiben und Lesen eines MPI – Programms auf Basis der native PVFS – API mit variierender Anzahl der I/O Server und variierender Anzahl der Rechenknoten im Cluster.

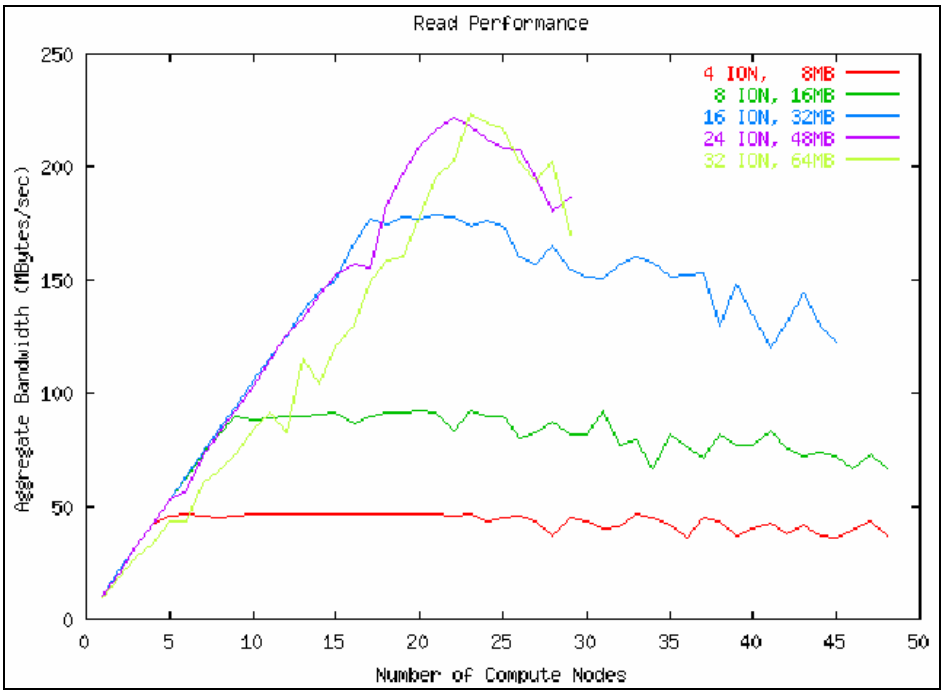

Abbildung 7: PVFS Lesegeschwindigkeit – Fast Ethernet (4 bis 32 I/O Server)

[Quelle: Carns 2000]

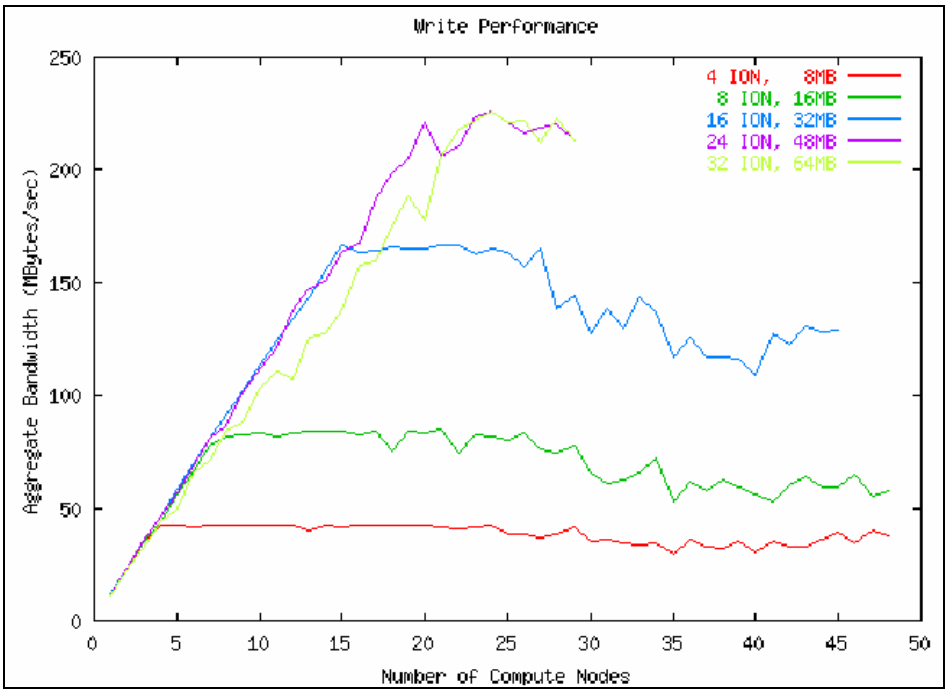

Abbildung 8: PVFS Schreibgeschwindigkeit – Fast Ethernet (4 bis 32 I/O Server)

[Quelle: Carns 2000]

Die Ergebnisse demonstrieren, dass mit steigender Anzahl von I/O Servern bis ca. 24 Stück und steigender Anzahl von Rechenknoten bis ebenso ca. 24 Stück, ein Ansteigen

der Bandbreite zu verzeichnen ist. Dabei kann die Lesegeschwindigkeit eines Rechenknoten von ca. 11 MByte pro Sekunde auf 222 MByte pro Sekunde bei 24 I/O Servern und 24 Clients gesteigert werden. Die Schreibgeschwindigkeit kann dabei sogar von ca. 10 MByte pro Sekunde bei einem Rechenknoten auf 226 MByte pro Sekunde mit 24 I/O Servern und 24 Clients aggregiert werden.

Myrinet

Entgegen des Fast Ethernet Netzwerks konnten durch Verknüpfung des Clusters anhand Myrinet höhere Geschwindigkeiten, gemäß folgenden Abbildungen, erzielt werden. Dabei wurde die Lesegeschwindigkeit eines Rechenknoten von 31 MByte pro Sekunde auf 687 MByte pro Sekunde mit 32 I/O Servern und 28 Rechenknoten gesteigert. Die Schreibgeschwindigkeit konnte ebenso von ca. 42 MByte pro Sekunde auf ca. 700 MByte pro Sekunde dank 32 I/O Server und 18 Clients gesteigert werden.

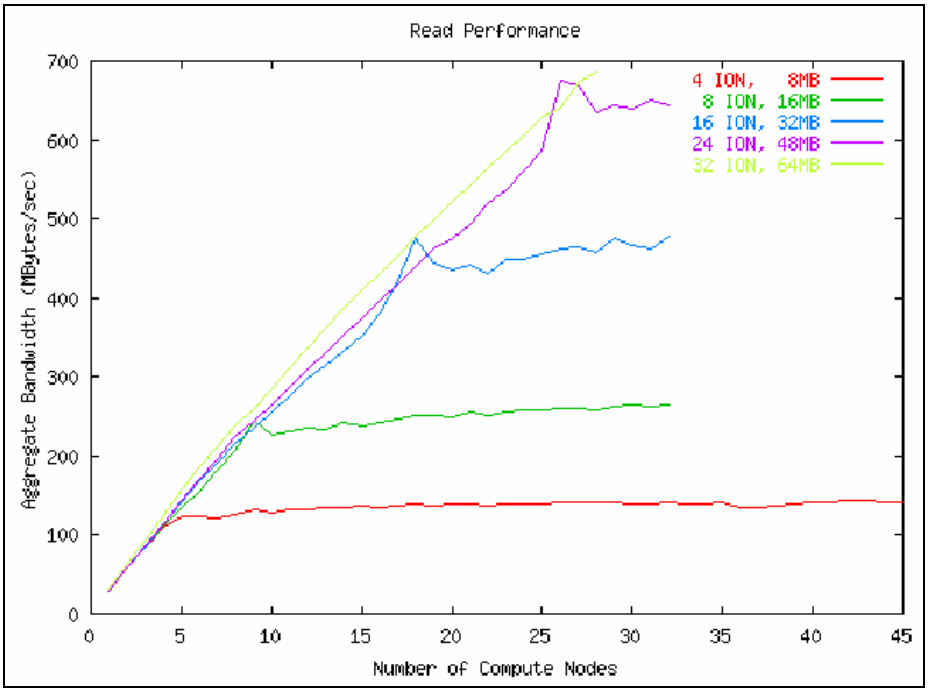

Abbildung 9: PVFS Lesegeschwindigkeit – Myrinet (4 bis 32 I/O Server)

[Quelle: Carns 2000]

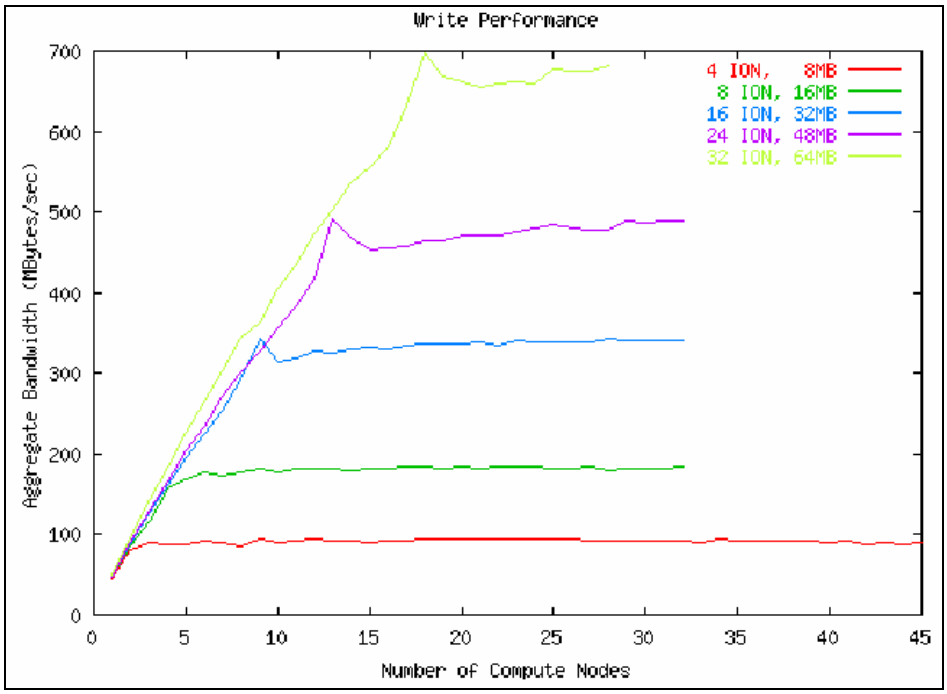

Abbildung 10: PVFS Schreibgeschwindigkeit – Myrinet (4 bis 32 I/O Server)

[Quelle: Carns 2000]

2.6.2 Simultanes Lesen und Schreiben mit native PVFS und MPI-IO

Für den Vergleich zwischen einer Programmimplementierung anhand native PVFS und einer durch ROMIO MPI-IO wurden die Messungen mit einer festen Anzahl von 32 I/O Servern und einer variablen Anzahl von Rechenknoten vorgenommen. Abbildung 11 zeigt, dass die native PVFS Implementierung bei Lese- und Schreibgeschwindigkeitsmessungen stets 7 bis 8 % besser abschneidet als die MPI-IO Implementierung. Das Testteam führte dies auf den Overhead der ROMIO Implementierung durch den Portabilitätsvorteil von ROMIO zurück und erklärte, dass diese Prozentpunkte jedoch sicher über weitere Algorithmenoptimierungen herausgeholt werden könnten.

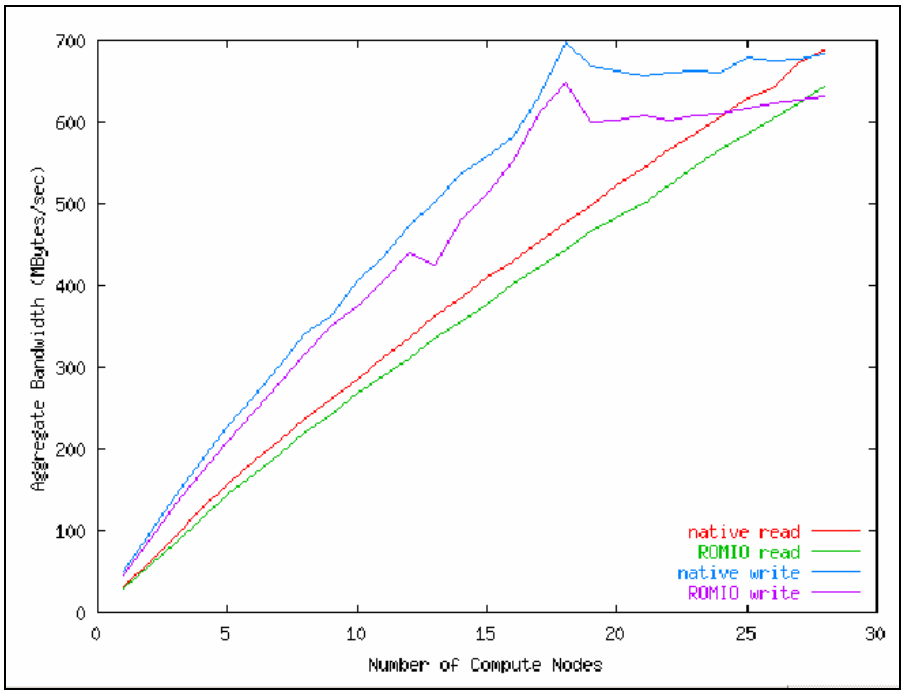

Abbildung 11: PVFS Performance – native PVFS vs. ROMIO MPI-IO Implementierung

[Quelle: Carns 2000]

2.6.3 Performanceanalysen

Auf Basis der Messungen und Betrachtungen wird deutlich, welche charakteristischen Merkmale ein PVFS System in einem Cluster aufweist. Die aggregierte Bandbreite ist unmittelbar von der Anzahl der verwendeten I/O – und Rechenknoten abhängig. Außerdem können Netzperformance und die Zugriffsgeschwindigkeiten der verwendeten Speichermedien zum Flaschenhals werden, wie der Vergleich zwischen dem Fast Ethernet und dem Myrinet verbundenen Cluster aufzeigt. Die Bandbreitenbeschränkung der Speichermedien wird dabei mit steigender Anzahl an Rechenknoten zu einer oberen Schranke. In anderen Experimenten, mit einfachen PC Clustern, die in der Summe aus weniger als 32 Knoten bestehen und kumulierte Hardwarekosten von < 10.000 US $ verursachten, konnten sogar Raten bis zu ca. 2 GByte Lesegeschwindigkeit und ca. 1,7 GByte Schreibgeschwindigkeit erzielt werden. [Bower-Cooley et al 2001; Ligon III 1999, S. 4-7]

3 Fazit zu PVFS – Version I und Ausblick zu Folgeentwicklungen

Die Ausführungen und Darstellungen in den Kapiteln 2.4 bis 2.6 verdeutlichen die Vor- und Nachteile eines parallelen Dateisystems auf Basis von PVFS I. In den meisten Fällen können an die identifizierten Vorteile auch Nachteile angeknüpft werden, sodass die PVFS Version I einen weiten Raum für Optimierungen lässt. [Argonne 2005; Carns 2000; Carns 2002; Ligon III 1999, S. 2f, 7]

- Vorteil stellt beispielsweise der transparente Zugriff für Benutzer dar, die das Dateisystem nach Installation und Start wie ein gewöhnliches Dateisystem verwenden können, sodass sie sich nicht um beispielsweise Datenverteilung und –blockbildung kümmern müssen, dies jedoch bei Bedarf können.
- In der Version I entspricht Datenverteilung dem RAID – Level 0 Konzept, was jedoch keine Redundanzen in den Daten zwecks Ausfall- und Datensicherung beinhaltet. Dieser Punkt ist daher einer der Aspekte, die in zukünftigen PVFS Versionen ausgeführt und implementiert werden könnten. Ferner erlauben die Datenverteilungsmechanismen, die von den Rechenknoten zugegriffenen Daten über viele verschiedene Wege im Cluster zu erreichen und sorgen auf diese Weise präventiv für eine Lastverteilung, sodass nicht einzelne Speichergeräte zum Flaschenhals der Applikation werden. Dieser Ansatz erlaubt demnach die Aggregierung der Einzelbandbreiten und realisiert somit außerordentlich hohe Geschwindigkeiten.
- Aus vorigem Punkt wird deutlich, dass es sich bei dem PVFS Konzept um ein Modell handelt, dass es erlaubt hohe Bandbreiten auf vergleichsweise günstigen Hardwarekomponenten im Cluster zu erzielen. Diese preiswerte Variante, die in beispielhaften Experimenten von Ligon III 1999 bis zu 2 GByte – Bandbreiten erreichte, bildet daher eine Alternativlösung für die Anwendung in Industrie und Wissenschaft.
- Die verschiedenartigen Verwendungskonzepte bilden einen weiteren Vorteil von PVFS I. Auf diese Weise können existierende Programme die Plattform verwenden oder Neuimplementierungen auf den portablen Standard aus der Parallelprogrammierung: ROMIO MPI-IO aufsetzen. Insbesondere der MPI – Ansatz ermöglicht trotz seiner Plattformunabhängigkeit auf Kosten eines konstanten Overheads, reichhaltige Optimierungsmöglichkeiten durch die Integration verschiedenster Operationen in den MPI-2 – Standard. Eben diese Kooperation fördert darüber hinaus die Popularität des Systems durch die tiefe Integration im de facto Standard der Parallelprogrammierung. Andererseits darf jedoch trotz der Verwendung von

Standards nicht die Komplexität der Programmierung solcher Systeme unterschätzt werden. Insbesondere wenn nicht der MPI Ansatz, sondern die beiden anderen Ansätze zur Verwendung von PVFS verwendet werden, gestaltet sich die Adaption verschiedener Architekturen, Systeme und Programmierwerkzeuge als schwierige Aufgabe.

- Die Verwendung von TCP zur Implementation unterstützt die Unabhängigkeit von proprietären Kommunikationsprotokollen, sodass die Plattformunabhängigkeit wesentlich unterstützt wird. Jedoch besteht die Gefahr, dass die Protokollperformance zum Flaschenhals wird, wenn Aspekte eines anderen Anwendungsgebietes in dem Protokoll implementiert sind, wie beispielsweise der „Slow – Start" – Mechanismus bei TCP, und somit bei sehr schnellen Netzen die dadurch zur Verfügung stehende Performance nicht ad hoc genutzt werden kann.
- Trotz aller Chancen und Kostenvorteile, die durch die PVFS Implementierung ermöglicht werden, betrifft die Wartbarkeit und Benutzbarkeit eines solchen Systems zahlreiche Anwender, sodass dieser Aspekt eine sehr relevante Rolle bezüglich des Systemeinsatzes und der Systempopularität spielt. Fortführungen der Entwicklung müssen sich demnach unabdingbar mit den Gesichtspunkten Installation, Konfiguration und Administration komplexer PVFS Cluster beschäftigen.

Entsprechend dieser anschaulichen Gesichtspunkte sind Fortentwicklungen an dem Dateisystem unentbehrlich. Wesentliche Rollen werden dabei die Punkte Skalierbarkeit, Zuverlässigkeit, Betriebssicherheit und Schnittstellenkompatibilität bilden, um PVFS zukünftig als günstige Alternative im industriellen Umfeld zu etablieren und für große Cluster mit mehreren 100 Knoten brauchbar zu gestalten.

Literaturverzeichnis

[Argonne 2005] Argonne National Laboratory (Mathematics and Computer Science Division) & Clemson University (Parallel Architecture Research Laboratory) [Editors]: The Parallel Virtual File System - Description. Clemson, 2005. http://www.parl.clemson.edu/pvfs/desc.html (06.04.2005)

[Bower-Cooley et al 2001] Bower-Cooley, J.; Guchereau, J.; Mache, J.; Thomas, P.; Wilkinson, M.: How to achieve 1 GByte/sec I/O throughput with commodity IDE disks. Proceedings of SC2001 - 14th ACM/ IEEE Conference on High-Performance Networking and Computing, Portland, 2001. http://www.lclark.edu/~jmache/sc2001.html (06.04.2005)

[Carns 2000] Carns, P. H.; Ligon III, W. B.; Ross, R. B.; Thakur, R.: PVFS: A Parallel File System For Linux Clusters. Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, 2000 http://www.parl.clemson.edu/pvfs/el2000/extreme2000.html (18.04.2005)

[Carns 2002] Carns, P. H.; Latham, R.; Ligon III, W. B.; Ross, R. B.: Using the Parallel Virtual File System. The Parallel Virtual File System – User Guide, Clemson, 2002. http://www.parl.clemson.edu/pvfs/user-guide.html (06.04.2005)

[Ligon III 1996] Ligon III, W. B.; Ross, R. B.: Implementation and Performance of a Parallel File System for High Performance Distributed Applications. Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing, Clemson, 1996. http://www.parl.clemson.edu/pvfs/hpdc96/hpdc96.html (06.04.2005)

[Ligon III 1999] Ligon III, W. B.; Ross, R. B.: An Overview of the Parallel Virtual File System. Proceedings of the 1999 Extreme Linux Workshop, Clemson, 1999. http://www.parl.clemson.edu/pvfs/extreme99/extreme-pvfs.ps (06.04.2005)

[PVFS 2005] Parallel Architecture Research Laboratory (Hrsg.): The parallel Virtual File System, Clemson University, http://www.parl.clemson.edu/pvfs/desc.html (18.04.2005)

[Rajaram 2002] Rajaram, K.: Principal Design Criteria Influencing the Performance of a Portable, High Performance Parallel I/O Implementation. Master's Thesis, Mississippi State University – Department of Computer Science, Mississippi 2002. http://sun.library.msstate.edu/ETD-db/theses/available/etd-04052002-105711/unrestricted/thesis.pdf (06.04.2005)

[Schütt 2001] Schütt, T: PVFS – Parallel Virtual File System, TU-Berlin, Berlin, 2001, http://www.zib.de/schintke/Vorlesungen/Datamngmnt/ausarbeitung_thorsten_pvfs.pdf (07.05.2005)