Fachhochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich Informatik
Department of Computer Science

Parallel Systems

Prof. Dr. Rudolf Berrendorf

Seminararbeit im Masterstudiengang über

Parallel Virtual File System

Version 2

am 24. Mai 2005

von

Marco Krause und Amir Napelyani

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

Abb.	Abbildung
API	Application Programming Interface
BMI	Buffered Message Interface
D.h.	Das heißt
FS	File System
ID	Identifikation
I/O	Input / Output
IP	Internet Protocol
MPI	Message Passing Interface
NFS	Network File System
POSIX	Portable Operating System Interface for UNIX
PVFS	Parallel Virtual File System
RAID	Redundant Arrays of Inexpensive Discs
TCP	Transmission Control Protocol
VFS	Virtual File System
z.B.	zum Beispiel

1 Einleitung

Parallel Dateisysteme dienen der effizienten Nutzung und Verwaltung großer Datenmengen in Hochleistungssystemen. Seit Mitte der 1990-er Jahre ist PVFS1 bei Linux-Clustern Der erfolgreichste Ansatz hierfür. Nach der Entwicklung von PVFS1 haben sich die Entwickler dazu entschlossen, das System komplett zu überarbeiten und haben vor kurzem eine erste Version von PVFS2 veröffentlicht.
Diese Seminararbeit soll die wichtigsten Eigenschaften von PVFS2 erläutern.

1.1 Erklärung Parallel Virtual File System

Um in großen Clustersystemen möglichst effizient auf die bis zu Petabyte großen Speicher zuzugreifen, hat man den Ansatz des globalen Dateisystems (Abb. 1) aufgegeben und ist zu parallelen verteilten Dateisystemen (Abb. 2) übergegangen, wodurch die verfügbare Speicherbandbreite vergrößert wurde.

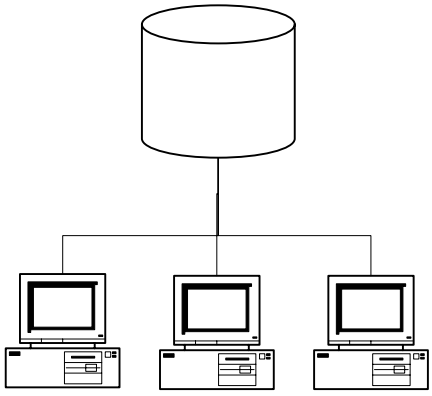

Abb. 1 - Globales Dateisystem

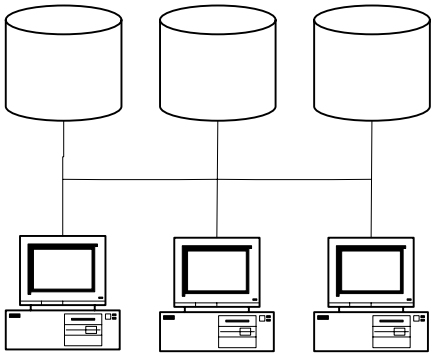

Abb. 2 - Paralleles verteiltes Dateisystem

Bei PVFS (Parallel Virtual File System) handelt es sich um ein paralleles virtuelles Dateisystem. Das heißt, PVFS ist ein paralleles verteiltes Dateisystem, welches Daten über das gesamte System verteilt abspeichert und parallele Zugriffe auf diese Dateien koordiniert, und ein virtuelles Dateisystem, welches nicht selber über ein Dateisystem verfügt, sondern auf ein bestehendes Dateisystem aufbaut.

1.2 Rückblick Parallel Virtual File System 1

PVFS1 ist bisher das erfolgreichste parallele virtuelle Dateisystem auf Linux-Clustern und wird sowohl in wissenschaftliche Rechenzentren als auch in der Forschung eingesetzt. Trotz des Erfolgs ließ PVFS1 noch einen weiten Spielraum für Verbesserungen offen. So lies Version 1 z.B. keine Redundanz der Daten bezüglich Ausfallsicherheit der Server zu und der Verteilungsmechanismus der Daten konnte zum Flaschenhals der Performancefähigkeit werden. Folglich wurde versucht die Nachteile der Version 1 durch die folgende Optimierung zu vermindern [Adam Staron 2005].

1.3 Rückblick Optimierung von Parallel Virtual File System 1

Nachdem PVFS1 noch Platz für Verbesserungen lies, wurden in der Optimierung verschiedene Ansätze eingeführt, die eine Performancesteigerung zwischen 30% und 70% beim Einsatz von PVFS1 ermöglichten. Der Unifier wurde entwickelt, der die Leistung der Serveranwendungen verbesserte, indem er z.B. überflüssige Daten im I/O-Pfad beseitigte und den Speicherregistrierungs- und Deregistrationsaufwand reduzierte. Weiterhin wurden Möglichkeiten aufgezeigt, wie die I/O-Performance optimiert werden konnten.
Jedoch wurden nicht alle Nachteile der Version 1 behoben, sodass schließlich eine Version 2 entwickelt wurde [Lau Wever 2005].

1.4 Gründe für die Entwicklung von Parallel Virtual File System 2

PVFS1 musste nach Ansicht der Entwickler aus mehreren Gründen weiterentwickelt werden. Zum Einen musste der Sourcecode überarbeitet werden, da er im Laufe der Zeit zu chaotisch wurde. Man implementierte immer mehr Funktionalitäten und nutze PVFS1 für Einsatzgebiete, an die man bei der Entwicklung nicht dachte. Des Weiteren kamen die Entwickler, durch neue Erfahrungen, zu der Einsicht, dass der Kern von PVFS1 nicht mehr angemessen für das Einsatzgebiet paralleler Dateisysteme sei. Sie erkannten, wie Anwendungen dieses Dateisystem nutzen und wie die zugrunde liegende Hardware besser eingebunden werden konnte. PVFS2 sollte ein robustes Hochleistungs-Dateisystem werden. Weitere Gründe für die Entwicklung von PVFS2 waren, dass die Entwickler nicht länger nur Workarounds für Probleme, die sich aus dem Softwaredesign ergaben, ausarbeiten wollten und vor allem, dass sie sich durch das Design behindert fühlten. PVFS1

war ihnen zu Socketlastig und unterstützte keine heterogenen Systeme mit unterschiedlichen endian Darstellungen.
Folglich war für die Entwickler der logischste Schritt die Weiterentwicklung von PVFS1.

1.5 Unterschiede zu PVFS1

Bei PVFS1 handelt es sich nicht um eine reine Weiterentwicklung von PVFS1, sondern um eine komplette Neugestaltung. Der Sourcecode wurde von Grund auf neu geschrieben.
Einige wichtige Features dieses neuen Designs sind:

- Modularer Netzwerkbetrieb und Speicher
- Flexible Datenverteilung
- Verteilte Metadaten
- Statusfreie Server und Clients
- Unterstützung der gleichzeitigen Verarbeitung
- Flexibles mappen von Dateireferenzen zu Servern
- Redundanz von Daten und Metadaten

Modularer Netzwerkbetrieb und Speicher

Die meisten Clustersysteme heutzutage nutzen eine Vielzahl verschiedener Netzwerktechnologien und es werden in der Zukunft weitere neue Technologien entstehen. Diese ganzen verschiedenen Netzwerktechnologien zu unterstützen kann zukünftig sehr entscheidend sein.
In der Speichertechnologie sieht es ebenso aus, es werden viele verschiedene Technologien benutzt. Auch hier müssen alle Technologien unterstütz werden.
Daher haben sich die Entwickler dazu entschlossen in PVFS2 BMI und Trove, ein Speicherinterface, zu nutzen, die verschiedene API´s für Speicher und Netzwerk anbieten (Abb. 3).

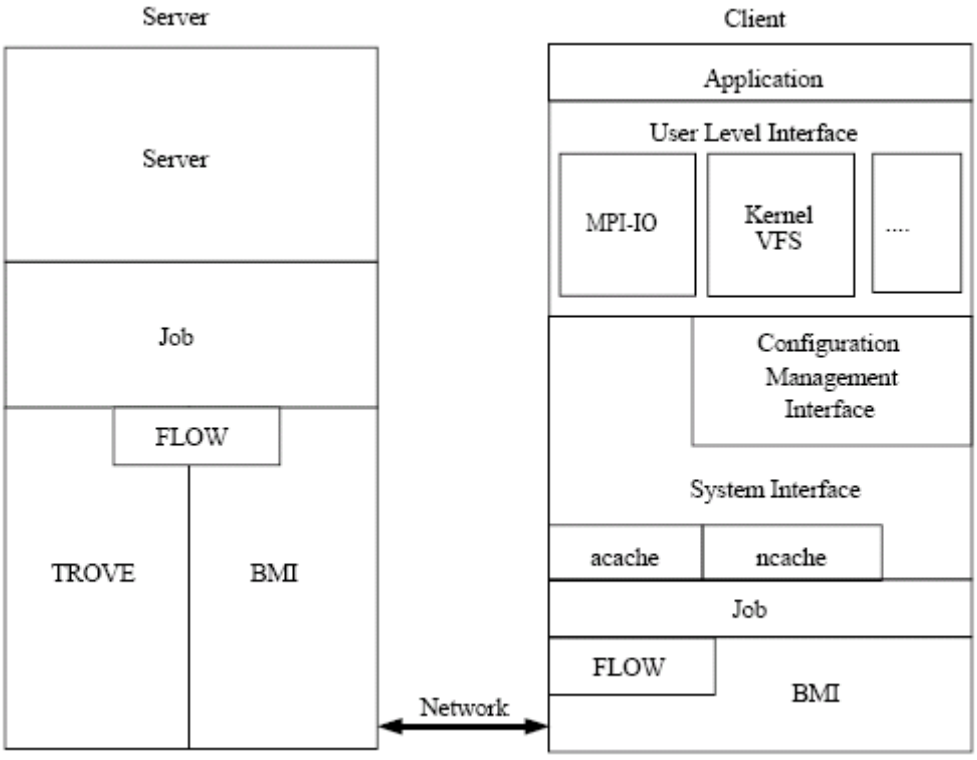

Abb. 3 - Aufbau PVFS2 [Sadleder 2004]

Verteilte Metadaten

In PVFS1 gab es nur einen Metadatenserver. Daraus ergaben sich zwei Nachteile; es existierte eine erhöhte Fehleranfälligkeit („Singlepoint of Error“) und der Server stellte im Bezug auf die Performance einen Flaschenhals dar. In der Praxis, war der Metadatenserver jedoch selten der Flaschenhals bei großen parallelen Anwendungen, da die meisten Operationen I/O-Operationen waren und der Metadatenserver dort nicht beteiligt war. Dennoch entschied man sich dazu, einen solchen Singlepoint of Contact zu vermeiden.
In PVFS2 gibt es daher mehrere Metadatenserver, die es ermöglichen die Metadaten verschieden Datei auf verschiedenen Servern ablegen zu können, wodurch sich Anwendungen die auf verschiedene Daten zugreifen wollen, weniger gegenseitig beeinflussen.

Zustandslose Server und Clients

Ebenso, wie mit der Anzahl beteiligter Einheiten das System wächst, erhöhen sich auch Fehlerwahrscheinlichkeiten. Es muss folglich alles getan werden um diese Wahrscheinlichkeiten so niedrig wie möglich zu halten.
Ein Ansatz hierfür, der bereits beim Network File System (NFS) angewandt wird, ist der Verzicht auf Zustände und Locks. D.h. verschwindet, z.B. durch einen Absturz, aus dem Netzwerk ein Client, kann der Server immer noch die Dateien anderen Clients zur Verfü-

gung stellen, ohne das System erst durch mühsame Recovery-Operationen wieder arbeitsfähig zu machen.
PVFS2 verfolgt den gleichen Ansatz; es ist ein zustandsloses System und verwendet keine Locks als Teil einer Client-Server Interaktion. Dadurch wird der Recoveryprozess im Fehlerfall erheblich vereinfacht und ermöglicht so eine standard Hochverfügbarkeitslösung für eine Ausfallsicherung des Servers.

Unterstützung der gleichzeitigen Verarbeitung

Für parallele Systeme ist die gleichzeitige Verarbeitung ein kritischer Faktor für das System. PVFS2-Server und -Clients sind rund um einen Zustandsautomaten konzipiert, der fest mit einer Komponente verkoppelt ist, die über das gesamte System hinweg überwacht, in wie fern Operationen beendet wurden. Weiterhin werden dort, wo es nötig ist, Threads benutzt, um den nicht blockierenden Zugriff aller Geräte zu gewährleisten. Diese Kombination aus Threads, Zustandsautomaten und Benachrichtigungen über beendete Operationen, erlaubt es einen möglichen Fortschritt bestimmter Berechnungen schnell zu identifizieren und vermeidet die Serialisierung unabhängiger Operationen, innerhalb der Server und Clients.
Dieses Design hat den Nebeneffekt, dass es eine Unterstützung auf unterster Ebene für Clientseitige asynchrone Operationen ermöglicht. Durch diese Unterstützung können nicht blockierende Operationen, unter MPI-IO, leicht implementiert und und vorteilhaft genutzt werden.

Flexibles Mappen von Dateireferenzen zu Servern

In parallelen Dateisystemen kann das Mappen der Dateireferenzen auf ihren Speicherstellen auf den jeweiligen Geräten entweder bei der Rekonfiguration des Dateisystems helfen oder hinderlich sein.
In PVFS2 werden alle Dateien in so genannte *Datafiles* aufgeteilt und jedes dieser Datafiles besitzt eine eigene Referenz. Welche Datafiles auf welchen Servern liegen, erfährt ein Client durch eine Tabelle, die er während der Konfiguration lädt. Wird ein Server dem System hinzugefügt werden, geschieht dies, indem ein neuer Referenzbereich für den Server reserviert wird, alle Clients neu gestartet werden und so eine aktualisierte Tabelle bekommen. Sollen Server aus dem System herausgenommen werden, wird ähnlich verfahren. Zuerst werden alle Clients gestoppt, die Datafiles auf andere Server verteilt und schließlich alle Clients ebenfalls mit einer aktualisierten Tabelle neu gestartet.
Sobald grundlegenden Funktionalitäten stabil laufen, soll untersucht werden in, wie fern dies in einem laufenden System zu bewerkstelligen ist.

Redundanz von Daten und Metadaten

Ein weiterer Nachteil von PVFS1 war der mangelnde Support von Redundanz auf Serverebene. RAID Ansätze eignen sich, um Toleranz gegenüber Festplattenfehlern zu gewährleisten. Sollte jedoch ein Server komplett ausfallen, kann auf keine Dateien des Servers mehr zugegriffen werden, bis er wieder hergestellt wurde.

Eine Option wären herkömmliche Hochverfügbarkeitslösungen, die in PVFS2 für Metadaten- / Datenserver verwendet werden können. Jedoch wird hierbei einen shared Memory zwischen den beiden Servern benötigt, auf denen die Daten des Dateisystems gespeichert werden, sodass es für einige Nutzer unerschwinglich teuer wäre.

Eine andere Option, die derzeit noch untersucht wird, ist die so genannte *Lazy Redundancy*. Hierbei soll die Erzeugung redundanter Daten eine explizite Operation werden. Das erfordert einerseits vom User, dass dieser den konsistenten Zugriff sicherstellt. Andererseits ergeben sich dadurch einige Möglichkeiten für Optimierungen, wie das parallel Erzeugen redundanter Daten. Des Weiteren soll es dadurch möglich sein, berechnungsintensivere Algorithmen zu nutzen und eine größere Ausfallsicherheit zu ermöglichen.

Lazy Redundancy befindet sich derzeit noch in der konzeptuellen Phase, in der versucht wird sie ideal in das System einzubinden. Bis dahin bleiben die herkömmlichen Ausfallsicherheitslösungen in der aktuellen Version eingebunden.

2 Grundlagen PVFS2

2.1 Server

Während es in PVFS1 zwei verschiedene Servertypen gab, einer für die Metadaten und einer für die Daten des Dateisystems, gibt es in PVFS2 nur ein Servertyp, der nach wie vor ein UNIX Prozess ist, wodurch auf einem Knoten mehrere dieser Prozessoren laufen können. Durch eine Konfigurationsdatei kann anschließend jedem dieser Server eine Rolle im Dateisystem zugewiesen werden. Wie in PVFS1 sind diese Rollen Metadaten- und Datenserver, aber als Besonderheit kann jeder PVFS2-Server beide Rollen gleichzeitig wahrnehmen.

Jeder PVFS2-Server speichert die Daten für das parallele Dateisystem lokal ab. Der Speicher basiert auf UNIX Dateien, für Daten und auf einer Berkeley Datenbank für die Metadaten. Die genauen Spezifikationen stehen in der „Trove“ API.

2.2 Netzwerke

In PVFS2 besteht die Möglichkeit verschieden Netzwerktypen zu nutzen. Das ist durch eine Abstraktion möglich die Buffered Messaging Interface (BMI) genannt wird. Zurzeit unterstützt BMI TCP/IP, InfiniBand und Myrinet.

2.3 Schnittstellen

Zurzeit gibt es zwei Low-Level I/O-Interfaces, die von den Clients für den Zugriff auf das parallele Dateisystem genutzt werden. Zum Einen die UNIX API, die durch das Betriebssystem des Clients repräsentiert wird und zum Anderen das MPI-I/O Interface.

In PVFS1 existierte für den Zugriff ein ladbares Modul, das alle virtuellen Dateisystemoperationen in den User-Space exportierte, wo ein clientseitiger UNIX-Prozess mit den Servern interagiert hat. Später erledigte dies eine effizientere In-Kernel Version.

PVFS2 verfolgt einen ähnlichen Ansatz, wie der Ursprüngliche es getan hat. Ein ladbareres Kernelmodul exportiert Funktionen in den User-Space, wo ein UNIX-Prozess – der *pvfs2-client* – die Interaktion mit den Servern koordiniert. Die Entwickler haben diesen „Rückschritt“ von der In-Kernel Version getan, weil sie nicht genau wussten, ob sie vom Kernel aus vollständigen Zugriff auf die Netzwerk-API´s besitzen.

Die zweite API ist das MPI-IO Interface. Die ROMIO MPI-IO Implementierung für die Unterstützung der MPI-IO wurde von PVFS1 übernommen. D.h. ROMIO linkt für den Zugriff direkt zu einer low-level PVFS2 API, sodass Daten nicht durch das Betriebssystem verschoben und durch den *pvfs2-client* kommuniziert werden.

2.4 Client-Server Interaktionen

Zu Beginn der Kommunikation kontaktieren Clients jeden einzelnen Server und erhalten so Informationen über das Dateisystem. Erst nachdem ein Client diese Informationen besitzt, kann er auf dem PFVS2-System arbeiten.

Um Zugriff auf eine Datei zu erhalten, wird der Dateiname durch eine lookup-Operation in eine etwas unverständlichere Referenz oder ein *handle* aufgelöst. Besitzt ein Client den handle zu einer Datei, kann er versuchen auf irgendeine Region dieser Datei zugriff zu bekommen, wobei Freigabekontrollen fehlschlagen können. Wird ein handle ungültig, teilt der Server dies bei der nächsten Zugriffsanfrage mit.

Handles werden an späterer Stelle noch genauer beschrieben. Es sei nur kurz erwähnt, dass sie es uns ermöglichen, sie von einem beliebigen Prozess herauszugreifen, via MPI-Message zu einem anderen Prozess zuschicken, um sie dort für einen Zugriff auf die gleiche Datei zu benutzen. Dies ermöglicht den *MPI_File_open* Aufruf, durch einen einzelnen lookup und einem anschließenden Broadcast.

In PVFS2 gibt es auf den Servern keinen Status und keine Konzepte für „offene“ Dateien. Daraus ergeben sich jedoch einige Vorteile. Zum einen gibt es keinen shared Status, der verloren gehen kann, sollte ein Client oder Server ausfallen. Des Weiteren muss auch nichts gemacht werden, wenn eine Datei „geschlossen“ ist, außer um sie vom Server speichern zu lassen. In einem MPI-Programm kann das von einem einzelnen Prozess getan werden.

Für den PVFS2 Zugriff aus dem Betriebssystem heraus, gibt es weiterhin *Öffnen*- und *Schließen*-Funktionen, die ebenso gewohnt funktionieren wie *lseek*.

Es jedoch auch einige Nachteile. Einer ist der Umgang von UNIX mit ungelinkten offene Dateien. Bei lokalen Dateisystemen kann auf einmal geöffnet Dateien immer wieder zugegriffen. Um korrekt zu funktionieren, basieren bestimmte Programme auf diesem Vorgehen. In PVFS2 weiß man jedoch nicht, ob eine Datei geöffnet ist. Ist eine Datei also nicht gelinkt, ist sie weg.

2.5 Konsistenz aus der Clientsicht

Portable Operating System Interface (POSIX) ist ein entwickeltes standardisiertes Applikationsebeneninterface, das die Schnittstelle zwischen Applikation und dem Betriebssystem darstellt, das jedoch zu strikt für große parallele Dateisysteme ist. Verzichtet man auf die POSIX-Semantik, ergeben sich einige nützliche Möglichkeiten. Zudem unterstützen nur sehr wenige Dateisysteme POSIX. Daher unterstützt PVFS2 die POSIX-Semantik nicht.

PVFS2 unterstützt das atomare Schreiben auf nicht überschneidenden Regionen ebenso, wie auch auf nicht benachbarten, nicht überschneidenden Regionen. Das reicht aus, um

alle nichtatomaren Operationen für MPI-IO zu unterstützen. Die atomaren Operationen müssen auf einer höheren Ebene unterstützt werden. Dies wird wahrscheinlich eher in der Verbesserung von ROMIO umgesetzt werden als in der Dateisystem-Infrastruktur, die damit nur unnötig kompliziert wird. Es gibt folglich gute Gründe es in der MPI-IO Schicht umzusetzen als im Dateisystem.
Das Cachen der Verzeichnisstruktur ist in PVFS2 für eine konfigurierbare Dauer gestattet, die jedoch nach Möglichkeit auf null gesetzt werden sollte.

.

2.6 Konsistenz des Dateisystems

Eine der komplizierteren Dinge beim Aufbau eines verteilten Dateisystems ist die Bedeutung eines konsistenten Dateisystems bei konkurrieren Operationen insbesondere bei solchen, die die Verzeichnisstruktur verändern.
Herkömmliche verteilte Dateisysteme verwenden eine blockierende Infrastruktur, um Atomarität sicherzustellen. Diese blockierenden Systeme besitzen aber eine höhere Latenzzeit, sind häufig nur extrem kompliziert zu optimieren und benötigen eine gründliche Fehlerbehandlung. Da es zudem bisher weder Beweise von der Community für parallele I/O noch von der Community für verteilten Shared Memory gibt, dass diese blockierenden Systeme gut auf den hier betrachteten Clustersystemen laufen, hat man auf bei der Entwicklung von PVFS2 darauf verzichtet dieses System umzusetzen.
Stattdessen wurden Operationen, die die Dateisystem-Hierarchie verändern können, so genutzt, dass sie aus der Sicht des Dateisystems scheinbar atomar durchgeführt werden. Die Clients gehen dafür schrittweise, mit so genannten *Server Requests*, vor, sodass der Eindruck einer atomaren Operation entsteht. Für die Erstellung einer neuen Datei sind beispielsweise folgende Schritte nötig:

- Erzeugen eines Directory Eintrag
- Erzeugen eines Metadatenobjekt
- den Directory Eintrag auf das Metadatenobjekt verweisen
- Erzeugen von Datenobjekten um Daten für die neue Datei zu erhalten
- die Metadaten auf Datenobjekte verweisen

Werden diese Schritte in einer besonderen Reihenfolge ausgeführt, gelangt das Dateisystem in einen Zustand, in dem Verzeichniseintrag für eine Datei existiert, auf die nicht zugegriffen werden kann. Daher muss sorgfältig darauf geachtet werden, in welcher Abfolge die Operationen durchgeführt werden:

1. Neue Datenobjekte erzeugen, um Daten für die neue Datei zu bekommen,
2. ein neues Metadatenobjekt für die neue Datei erzeugen,
3. die Metadaten auf die Datenobjekte verweisen,

4. Ein neuen Verzeichniseintrag erzeugen und es auf das Metadatenobjekt verweisen.

Durch die Einhaltung dieser Reihenfolge bleibt das Dateisystem in einem konsistenten Zustand; die Datei existiert und es kann auf sie zugegriffen werden. Auf diese Weise werden alle PVFS2-Operationen durchgeführt.

Durch diesen Ansatz entsteht ein gewisser Grad an Komplexität; wenn der Prozess unterbrochen wird oder der letzte Schritt, das Erzeugen eines Verzeichnisses, fehlschlägt muss sehr viel bereinigt werden. Das Problem kann aber gelöst werden. Da diese Objekte nicht von anderen Objekten referenziert sind, können sie einfach gelöscht werden, ohne darauf zu achten, ob sie von anderen Prozessen benutzt werden.

3 Komponenten des Dateisystems

Ein PVFS2 Dateisystem kann aus den folgenden Komponenten bestehen (einige sind optional):

- pvfs2-server
- Systemschnittstelle
- Managementschnittstelle
- Linux Kernel Treiber
- pvfs2-client
- ROMIO PVFS2 Vorrichtung

Diese Komponenten, sind die Hauptbestandteile aus der Administrations- und Userperspektive.

Der pvfs2-server ist der Serverbestandteil des Dateisystems. Es läuft vollständig im Benutzerraum. Viele pvfs2-server Instanzen können auf beliebig viele unterschiedliche Maschinen laufen. Jede Instanz kann entweder als ein Metadata-Server, ein I/O Server oder als beides dienen (Abb. 4).

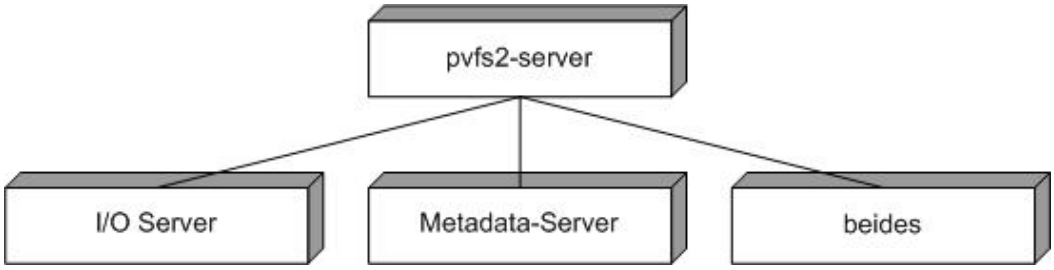

Abb. 4 - pvfs2-Server

Der Metadata-Server speichert Informationen über Dateien, wie z.B. Zugriffserlaubnis, Zeitstempel und Parameter über die Verteilung. Sie speichert aber auch die Verzeichnishierarchie ab. Die ersten Releasen PVFS2 unterstützen nur einen Metadata-Server pro Dateisystem, aber diese Beschränkung soll zukünftig aufgehoben werden.

I/O Server speichern die tatsächlichen Daten, die mit jeder Datei in Verbindung stehen. Typischerweise wird diese Verbindung mit Hilfe des Round Robins über mehrere Server realisiert.
Die Systemschnittstelle ist der Benutzerraum API auf der tiefsten Ebene, der Zugang zum PVFS2 Dateisystem liefert. Es war eigentlich nicht beabsichtigt die Systemschnittstelle als

ein Endbenutzer API zu realisieren; Anwendungsentwickler werden stattdessen angeregt, MPI-IO als ihre erste Wahl zu verwenden, oder für die Applikationsaufrufe die Standard Unix Aufrufe zu benutzen.
Die Systemschnittstelle wird als eine einzelne Bibliothek implementiert, genannt libpvfs2. Ihre API bildet nicht direkt alle POSIX-Funktionen ab. Insbesondere ist es eine zustandslose API, die kein Konzept von open(), close() oder Dateibeschreibungen hat. Diese API vereinfacht jedoch gleichzeitig den Task zur Kommunikation zwischen vielen Servern.

Die Managementschnittstelle ist eine ergänzende API zu der Systemschnittstelle, die Funktionalitäten hinzufügt, die normalerweise nicht für irgendwelche Dateisystembenutzer geeignet sind. Diese Funktionalitäten sind für Administratoren und für Applikationen, wie z.B. fsck oder Performance Monitoring gedacht, die Dateisysteminformationen aus der tiefen Ebene erfordern.

Der Linux Kernel Treiber ist ein Modul, das in einen unveränderten Linux Kernel geladen werden kann, um VFS - Unterstützung für PVFS2 anzubieten. Momentan ist dieses Feature nur für die Kernel Serie 2.6 implementiert. Sie agiert als der Bestandteil, der Standard Unixanwendungen (einschließlich Dienstprogramme wie ls und cp) erlaubt an PVFS2 zu arbeiten. Der Kernel-Treiber erfordert den Gebrauch einer Anwendung im Benutzerraum, genannt pvfs2-client.

Der pvfs2-client ist ein Dämon im Benutzerraum, dass die Kommunikation zwischen dem PVFS2 Server und dem Kernel Treiber ermöglicht. Seine Primärrolle ist, VFS-Operationen in Systemschnittstellen-Operationen umzuwandeln. Möchte ein Client durch die VFS-Schnittstelle Zugang zum Dateisystem haben, muss auf ihr ein pvfs2-client laufen.

Abb. 5 zeigt die Beziehungen zwischen den einzelnen Dateisystemkomponenten auf.

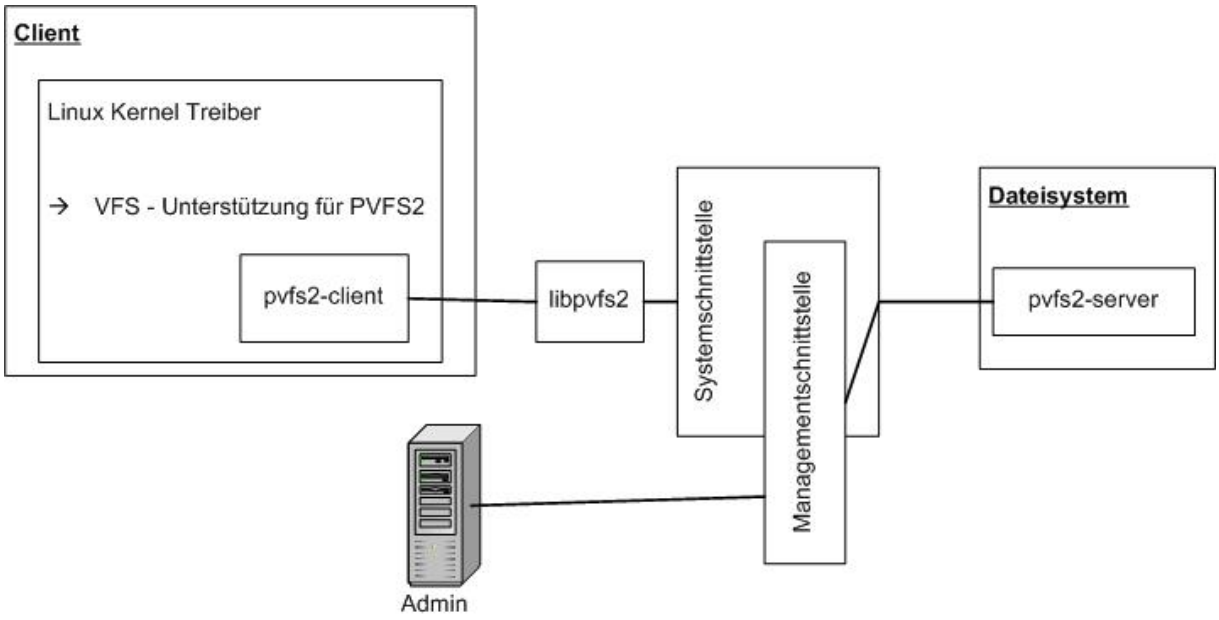

Abb. 5 - Abhängigkeit der Dateisystemkomponenten

Die ROMIO PVFS2 Vorrichtung ist ein Bestandteil der ROMIO MPI-IO Implementierung, dass MPI-IO Unterstützung für PVFS2 bietet. ROMIO ist in der Implementierung MPICH MPI bereits enthalten und schließt Treiber für einige Dateisysteme mit ein. Für mehr Details wird auf http://www.mcs.anl.gov/romio/ hingewiesen.

4 Handles

Handles sind spezielle digitale Erkennungszeichen für jedes Objekt, das auf dem PVFS2 gespeichert ist. Sie ermöglichen, dass das Betriebssystem diese Objekte anhand ihrer Handles finden und öffnen kann. Jede Datei, Ordner und symbolische Link besitzt so ein Erkennungszeichen. Zusätzlich werden verschiedene versteckte Objekte, die nicht von den Benutzern direkt manipuliert werden können, mit Handles dargestellt. Das sorgt für einen knappen, pfadunabhängigen Mechanismus für die Spezifikation, welche Objekte zur Kommunikation zwischen dem Client und dem Server angesprochen werden müssen. Wenn Dateisystemobjekte erstellt werden, generieren die Server automatisch neue Handles, sodass die Manipulation der Objekte durch den Benutzer für gewöhnlich nicht direkt stattfindet, sonder nur über die Handles.

Der zulässige Wertebereich, die Handles annehmen können, wird als „handle space“ bezeichnet.

Handle Bereiche

Handles sind im Wesentlichen sehr große Ganzzahlen. Dies bedeutet, dass wir den „handle space“ bequem in Teilmengen aufteilen können, indem wir einfach die Wertebereiche der Handles spezifizieren. Handle-Bereiche sind Gruppen von Handles, die durch die Wertebereiche, die sie enthalten können, beschrieben werden.

Um den “handle space” auf N Server zu verteilen, teilen wir den „handle space“ erst in N Handle-Bereiche und übertragen dann die Steuerung jedes Bereiches einem anderen Server (Abb. 6). Die Konfigurationsdateien des Dateisystems bieten ein Mechanismus an, dass die genaue Zuordnung zwischen Handle-Bereiche zum zugehörigem Server Hosts ermöglicht. Die Clients beeinflussen nur die Handle-Bereiche; die Zuordnung von den Bereichen zu den Servern ist unter einer Abstraktionsschicht versteckt. Dieses erlaubt eine größere Flexibilität und lässt zukünftige Eigenschaften wie transparente Migration (Betriebssystemwechsel) zu.

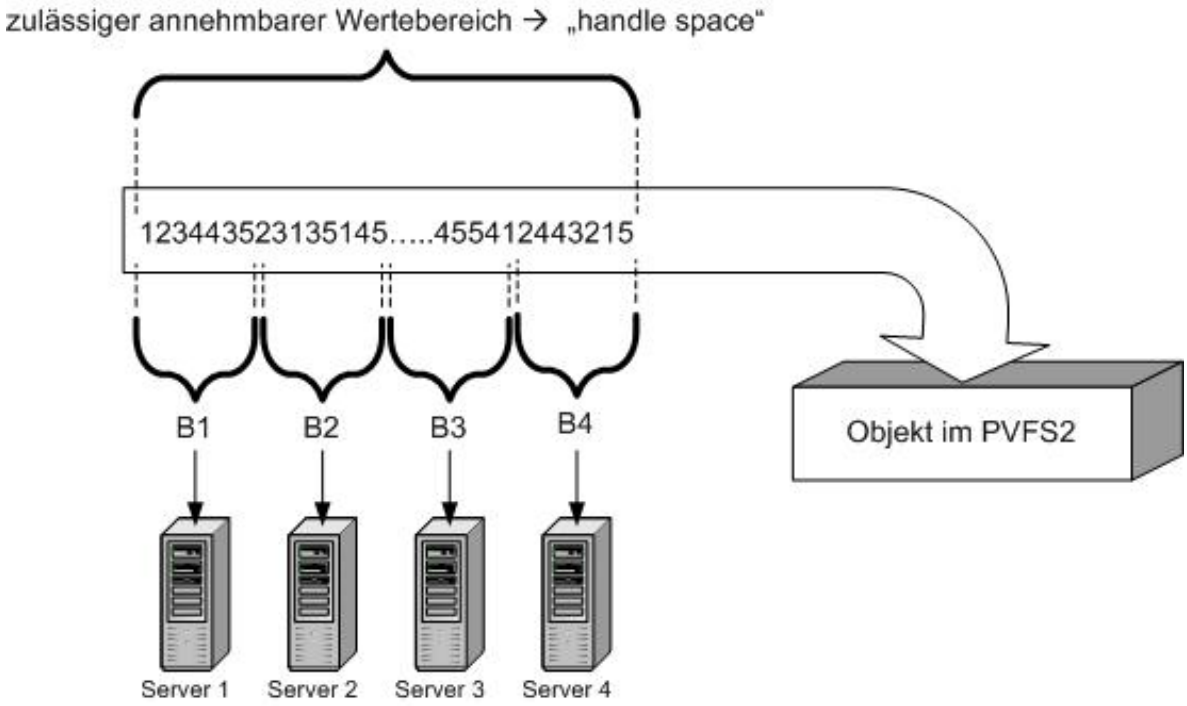

Abb. 6 - Handles

5 Dateisystem – Identifikation

Jedes PVFS2 Dateisystem, das durch einen bestimmten Server gehostet (bewirtet) wird, hat einen eindeutigen Bezeichner, der als Dateisystem ID oder „fs id“ (steht für: file system id) bekannt ist. Die ID des Dateisystems muss zur Erstellung des Dateisystems durch administrative Werkzeuge zugewiesen werden, damit sie quer über alle Server für ein gegebenes Dateisystem synchronisiert werden. Dateisysteme haben auch symbolische Namen, die in einem fs id durch Server aufgelöst werden können, um lesbarere Konfigurationsdateien zu produzieren.

Dateisystem IDs verweisen auch gelegentlich auf eine Ansammlung von IDs.

6 Installation auf einen Single Host

Dieser Abschnitt dokumentiert die Schritte, die notwendig sind um PVFS2 auf einem System zu konfigurieren, das als Client und Server für alle PVFS2 Operationen fungiert.

6.1 Server Konfiguration

Da es sich hierbei um eine Single Host Konfiguration handelt, muss nur ein Server-Dämon konfiguriert werden. Im ursprünglichen PVFS wurden die Metadata und I/O Server in zwei verschiedenen Programmen getrennt. PVFS2 hat jedoch nur einen einzigen Dämon, den pvfs2-server, welches beide Rollen übernimmt.

Der wichtigste Teil der Serverkonfiguration ist die Erzeugung der Konfigurationsdateien. Diese können unter Benutzung des pvfs2-genconfig Skriptes erzeugt werden. Es ist ein interaktives Skript, welches anhand einiger Fragen die gewünschte Konfiguration bestimmt.

Das pvfs2-genconfig Tool erzeugt zwei Konfigurationsdateien. Die Erste ist ein Dateisystem - Konfigurationsdatei, die für alle Server identisch ist. Die Zweite ist eine serverspezifische Konfigurationsdatei, die für jeden Server unterschiedlich ist. Die serverspezifischen Konfigurationsdateien beinhalten den Hostnamen des Servers, dass auch an den Dateien hinzugefügt wird. Das Ausführen des Skriptes sollte als Administrator durchgeführt werden, um die Konfigurationsdateien in ihrer Default /etc/ ablegen zu können.

Die erzeugten Konfigurationsdateien haben konservative Default Werte. Diese sind so ausgewählt, dass nach dem Erzeugen der Dateien ein reibungsloser Start garantiert wird.

6.2 Client Konfiguration

Es existieren zwei Methoden, um auf das PVFS2 Dateisystem zu zugreifen. Durch die Benutzung des Kernel-Moduls wird die Standard Linux Dateisystemkompatibilität zur Verfügung gestellt. Diese Schnittstelle erlaubt dem Benutzer, vorhandene Binärcodes und System Dienstprogramme ohne Neuübersetzung zu starten. Die zweite Möglichkeit um Zugriff auf das PVFS2 Dateisystem zu erlangen, ist der Zugriff durch die MPI-IO Schnittstelle. Sie ist auf das libpvfs2 Bibliothek aufgebaut und lässt höhere Leistung für parallele Anwendungen zu.

7 Installation auf einem Cluster

Es ist von großer Bedeutung, die einzelnen Rollen der Server (Knoten), die sie im PVFS2 spielen, in Hintergedanken zu halten. Folgende Rollen sind möglich: Metadata-Server, I/O Server oder Client

Ein Metadata-Server ist ein Knoten, dass die Metadaten (wie z.B.: Zugriffserlaubnis und Zeitstempel) für das Dateisystem aufrecht hält. Die eigentliche Speicherung einer Teilmenge der PVFS2 Daten übernimmt der I/O Server. Der Client ist in der Lage PVFS2 Dateien zu Lesen bzw. zu Schreiben. Typischerweise laufen die Applikationen deshalb auf einem PVFS2 Client. So wird der direkte Zugriff auf das Dateisystem ermöglicht.

Eine Maschine kann gleichzeitig eine, zwei oder all diese Rollen besetzen. Anders als PVFS1, erfordert jede Rolle nur den binären pfvs2-server. Bei einem Neustart konsultiert sie die clusterweite- und die knotenspezifische Konfigurationsdatei, um zu erfahren, welche Rolle der pvfs2-server auf dieser Maschine annehmen soll.

Zurzeit wird nur ein Metadata-Server unterstützt. Es kann aber viele I/O Server und Clients geben. In diesem Abschnitt werden die Bestandteile und die Konfigurationsdateien, die benötigt werden, um jede Rolle zu erfüllen beschrieben.

Server Konfiguration

Als Basis wir davon ausgegangen, dass auf jedem Knoten eine „make install“ ausgeführt worden ist, oder jemand dafür gesorgt hat, dass die pvfs2 ausführbaren Dateien, Header und Bibliotheken mit anderen Mitteln für die Maschinen zur Verfügung stehen.

Die Installation von PVFS2 auf einem Cluster ist der Installation auf einem Single Host ziemlich ähnlich. Es muss ein Master Konfigurationsdatei und n kleinere knotenspezifische Konfigurationsdateien generiert werden.
Die Konfigurationsdateien müssen jetzt an alle Server-Knoten verteilt werden. Falls die zur Verfügung gestellten rc Skripte (Redhat style) benutzt werden, kann man alle Konfigurationsdateien an alle Knoten verteilen und jeder Server würde beim Hochfahren die korrekte Konfigurationsdatei, versehen mit seinem Hostnamen, herauspicken (Abb. 7).

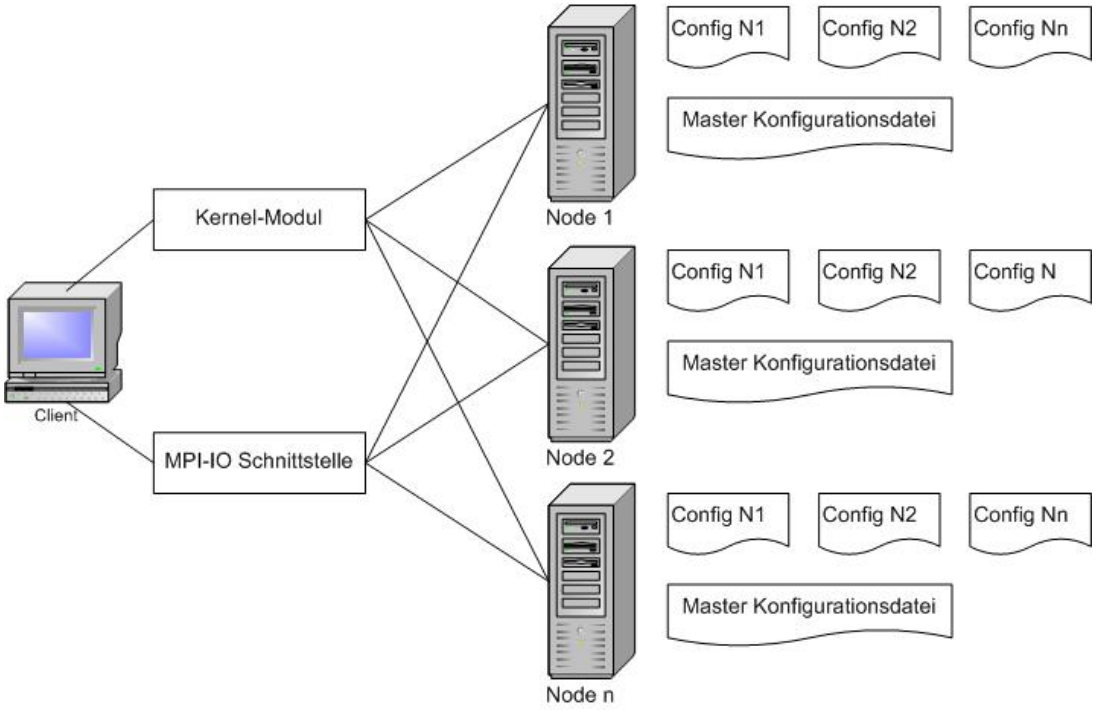

Abb. 7 - Installation auf einem Cluster

Die Client Installation ist ähnlich der Installation des Clients auf einem Single Host.

Literaturverzeichnis

[Adam Staron 2005] Adam, B., Staron, C.: Parallel Virtual File System (PVFS) I - Architektur, Verwendung und Eigenschaften. Sankt Augustin, 2005.

[Lau Wever 2005] Lau, I., L., Wever, P.: Optimierungsmöglichkeiten für PVFS1. Sankt Augustin 2005.

[PVFS2 2005] PVFS Homepage. Online http://www.pvfs.org/pvfs2/, 19.06.2005.

[Sadleder 2004] Sadleder, P.: Änderung der Datenverteilungsfunktion im parallele Dateisystem PVFS2. Online archiv.ub.uni-heidelberg.de/ volltextserver/volltexte/2005/55 24/, 19.06.2005,