Fachhochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich Informatik
Department of Computer Science

GPFS General Parallel File System

im Studiengang

Master of Science in Computer Science
SS 2005

Von

Numiel Ghebre und Felix Rommel

Betreuer: Prof. Dr. Berrendorf

Inhaltsverzeichnis

1 Einleitung

Manche Probleme lassen sich nicht mit einem einzelnen Rechner lösen. Auch mit heutigen Mehrprozessor-Maschinen können nicht alle Aufgaben gelöst werden, so lange diese einzeln eingesetzt werden. Erst der Zusammenschluss vieler Maschinen zu Clustern bringt oft die gewünschten Ergebnisse.

Cluster können beliebig erweitert werden - sei es Rechen- oder Plattenkapazität - indem neue Knoten hinzugefügt werden. Da sie aus unabhängigen und redundanten Computern bestehen sind sie fehlerredundant. Eine Folge davon ist ein stetig wachsendes Interesse an der Clustertechnologie.

Um Cluster voll nutzen zu können müssen Programme parallelisiert werden. Dies wird durch Partitionierung erreicht, d. h. mehrere Tasks teilen sich die im Cluster zur Verfügung stehenden Ressourcen, wie Festplatten-Speicher. Um die Volle Leistungsfähigkeit eines Clusters zu erreichen muss ein paralleles Dateisystem wie GPFS zum Einsatz kommen.

GPFS wurde so konzipiert, dass damit hunderte von Knoten betrieben werden können. Einige der größten Rechenanlagen der Welt verwenden heute GPFS, da es als das im Moment leistungsfähigste parallele Dateisystem betrachtet werden kann.

Kapitel 2 bietet eine allgemeine Übersicht über GPFS und zeigt die wichtigsten Eigenschaften auf. Im Kapitel 3 werden die technischen Details beleuchtet und ausgiebig dargestellt. Kapitel 4 beschäftigt sich mit der Fehlertoleranz des GPFS.

2 Übersicht

2.1 Herkunft GPFS

GPFS wird von IBM entwickelt. Es entstand aus dem Tiger-Shark-Multimedia-Dateisystem (vgl. [Hask1998]). In die Entwicklung sind viele Ideen der akademischen Gemeinschaft eingeflossen – wie z. B. das Distributed-Locking oder Wiederherstellungs-Technologien. Es ist unter anderem für die folgenden Plattformen verfügbar:

- GPFS für AIX 5L auf POWER™
- GPFS für Linux auf IBM AMD-Prozessor basierten Servern
- IBM eServer® xSeries®
- GPFS für Linux auf POWER

2.2 Was ist ein paralleles Dateisystem

Ein paralleles Dateisystem zeichnen folgende Eigenschaften aus (vgl. [ZaSi2003]):

- Verteilt Daten auf viele Datenträger
- Möglichst hoher E/A-Durchsatz steht im Vordergrund
- Meist optimiert für Anforderungen paralleler Anwendungen
- Nicht unbedingt geeignet für nicht-parallele Anwendungen
- Üblicherweise implementiert nach dem Client/Server Prinzip
- Compute-Nodes führen Client-Code aus
- E/A-Knoten führen Server-Code aus
- Es gibt aber auch Alternativen, wenn man auf Spezial Hardware zurückgreift

2.3 Besondere Anforderungen an ein PFS

Ein paralleles Dateisystem muss besondere Anforderungen erfüllen (vgl. [ZaSi2003]):

- Datenparallele Programmierung bedingt oft Zugriff auf eine Datei von vielen Prozessen aus
- Zugriff auf eine Datei wie auf ein Array
- Sowohl riesige Anzahl von Dateien als auch riesige Dateien.
- Zugriffsmuster müssen erkannt werden um Durchsatz zu erhöhen

2.4 Wichtige Eigenschaften des GPFS

Einige wichtige Eigenschaften des GPFS sind:

- GPFS ist POSIX-konform.
- Das GPFS liefert eine sehr gute Leistung beim sequentiellen Zugriff auf sehr große Dateien, sowohl seriell als auch parallel von verschiedenen Knoten aus. Dabei werden die Daten sowohl über die GPFS-Server als auch über die an den Servern

angeschlossenen Platten gestriped.

- Beim gleichzeitigen Zugriff von vielen Clients wird die Last über die vorhandenen Server verteilt. Der parallele, konsistente Zugriff auf eine einzige Datei - lesend und schreibend - ist mittels MPI-IO möglich.
- Eine weitere Stärke des GPFS ist die Ausfallsicherheit: Beim Ausfall eines Servers übernimmt ein anderer Server dessen Platten, so dass ein Benutzer oder ein Programm außer einer Leistungseinbuße nichts bemerkt.
- Das GPFS wird derzeit nur auf Hardware von IBM mit den Betriebssystemen AIX und Linux unterstützt. Zur schnellen Datenübertragung zwischen Client und Server kann entweder der SP Switch oder ein Fibre Channel (FC) Netzwerk genutzt werden.
- Verlangt homogene Systemlandschaft. Andere Systeme als AIX oder Linux können nur über DFS eingebunden werden.
- Das GPFS kann mit Recht als derzeit leistungsfähigstes paralleles Dateisystem bezeichnet werden; die Performance wird praktisch nur durch die Anzahl der eingesetzten Server und Plattensubsysteme begrenzt.
- Seit Version 2.3 können Knoten in laufendem Betrieb aus- und eingehängt werden ohne GPFS herunterfahren zu müssen.

2.5 GPFS Cluster und Knoten

Ein GPFS-Cluster besteht aus (vgl. [IBM22005], S. 3):

1. AIX 5L-Knoten, Linux-Knoten oder eine Kombination beider:
   - ein individuelles Dateisystem auf einem Computer in einem Cluster
   - Eine System-Partition welche ein Betriebssystem enthält. Einige pSeries-IBM-Systeme erlauben mehrere System-Partitionen auf einem Rechner. Jeder dieser System-Partitionen ist ein Knoten.
2. Network-Shared-Disks (NSD)
   Alle Platten oder Partitionen, die unter GPFS verwendet werden müssen NSD-Namen – Network -Shared-Disk - zugewiesen werden.
3. Ein einzelnes Netzwerk, ein LAN oder ein Switch wird für die GPFS-Kommunikation und NSD-Kommunikation verwendet.

2.6 GPFS-Shared-Disk-Architektur

GPFS geht davon aus, dass jeder Knoten Zugriff auf alle Festplatten hat. Dies kann unter Verwendung eines Storage-Area-Network (SAN), z. B. Fibre-Channel (FC) oder iSCSI ermöglicht werden (vgl. Abbildung 2).

Fibre Channel ist ein serieller Computer-Bus welcher für Verbindungen zwischen

Hochgeschwindigkeits-Geräten verwendet wird (vgl. [WiPe32005]). Besonders hervor zu heben ist hier die FC-Switched-Fabric (FC-SW) bei der alle Geräte durch FC-Switches verbunden sind und ein ähnliches Konzept wie Ethernet aufweist (vgl. Abbildung 1).

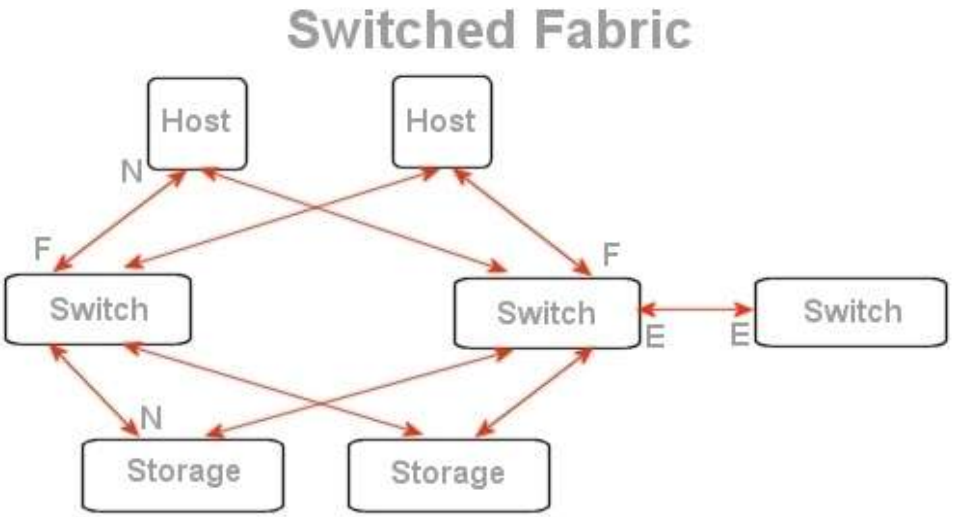

Abbildung 1Switched-Fabric (vgl. [WiPe22005])

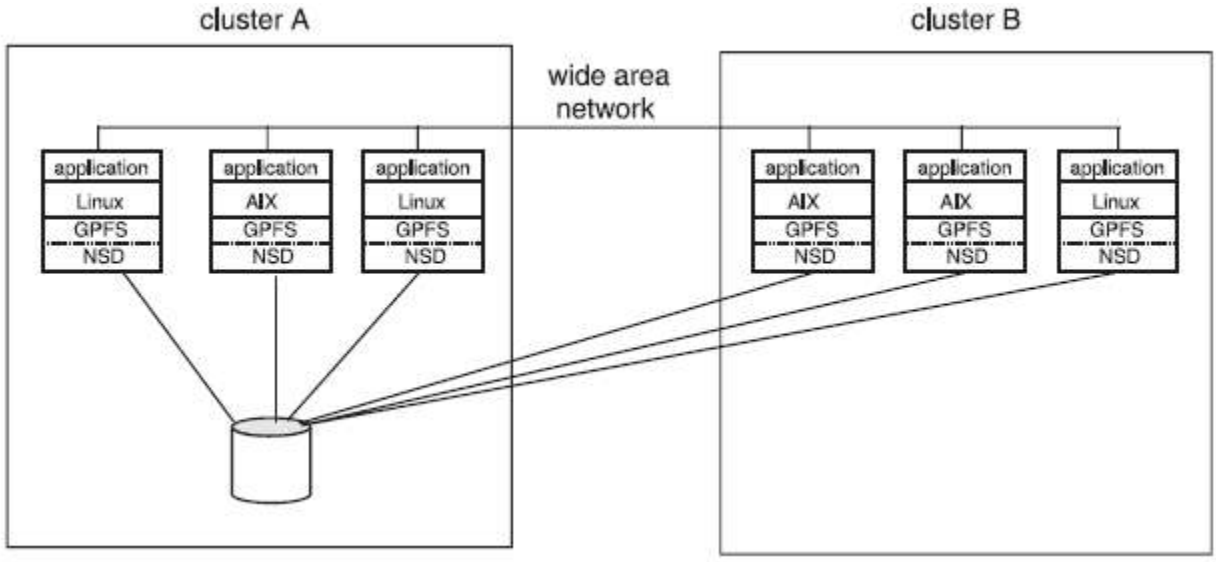

Abbildung 2: Entferntes einhängen eines Dateisystems unter Verwendung einer SAN-angehängten Platte (vgl. [IBM22002], S. xiv)

Eine andere Möglichkeit besteht darin, einzelne Platten an E/A-Server-Knoten zu hängen, mit deren Hilfe über eine Softwareschicht darauf zugegriffen werden kann. Die Kommunikation kann über ein Kommunikationsnetzwerk wie IBM's NSD realisiert werden (vgl. Abbildung 3). Es ist nicht entscheidend wie die gemeinsam verwendeten Platten implementiert sind – GPFS benötigt lediglich ein konventionelles Block-E/A-Interface ohne

intelligente Platten (vgl. [ScHa2002], Seite 3).

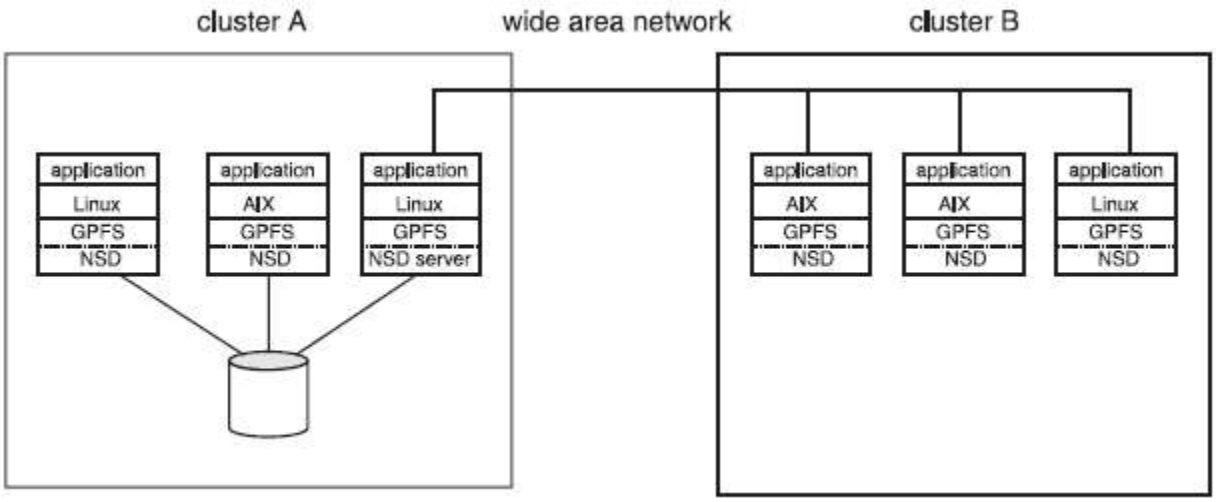

Abbildung 3: Entferntes einhängen eines Dateisystems unter Verwendung eines NSD-Server-Zugriffs (vgl. [IBM22002], S. xiv)

Falls mehrere Cluster auf ein Dateisystem zugreifen möchten, wird Open-Socket-Secure-Layer (OpenSSL) verwendet um Authentifizierungen für alle Netzwerkverbindungen durch zu führen. Daten, die über diese Leitungen verschickt werden, sind nicht geschützt.

3 Technische Eigenschaften

3.1 Arbeitsweise von GPFS

Eine grobe Übersicht über die Arbeitsweise von GPFS (vgl. [ZaSi2003], S. 24f):

- Zentraler Server ist Token Manager
- Verteilt Lock-Tocken um Schreibzugriffe abzusichern
- Verteilt Metanode-Token für verteilte Meta-Datenhaltung
- Solange nur wenige Prozesse auf eine Datei zugreifen, werden Lock-Token über Abschnitte der Datei vergeben
- Wenn viele Prozesse auf eine Datei zugreifen, werden die Datei-System-Knoten zu E/A-Knoten.
- Um einen allgemeineren Ansatz zu bieten, und trotzdem performant zu bleiben, werden hier also zwei Techniken gemischt.
- Ausfall-Sicherheit
- Daten-Replikation auf Dateisystem-Ebene

Im folgenden werden die einzelnen Komponenten und Eigenschaften von GPFS erläutert.

3.2 Der Konfigurations-Manager

Pro Cluster existiert ein Konfigurations-Manager (vgl. [IBM22005], S. 65). Er hat folgende Aufgaben:

- Platten-Wiederherstellung eines fehlerhaften Knotens in einem Cluster.
- Auswahl eines Knotens als Dateisystem-Manager. Der Konfigurations-Manager verhindert, dass mehrere Knoten die Rolle des Dateisystem-Managers übernehmen.

3.3 Der Dateisystem-Manager

Pro Dateisystem existiert ein Dateisystem-Manager welcher alle Tokens verwaltet. Die Dienste des Dateisystem-Managers werden im folgenden vorgestellt.

3.3.1 Dateisystem-Konfiguration

Der Dateisystem-Konfigurator verwaltet die folgenden Aufgaben (vgl. [IBM22005], S. 65):

- Platten hinzufügen
- Platten-Verfügbarkeit ändern.
- Dateisystem reparieren

Ein und Aushängen von Platten wird durch den Dateisystem-Manager und den anfragenden Knoten durchgeführt.

3.3.2 Verwaltung der Plattenplatz-Belegung

Der Konfigurations-Manager kontrolliert welche Regionen der Platten welchen Knoten zugeordnet werden. Dies erlaubt eine effektive, parallele Zuordnung von Speicherplatz.

3.3.3 Token-Verwaltung

Der Dateisystem-Server-Knoten führt die Dienste des Token-Managers-Servers aus (vgl. [IBM22005], S. 66).

Der Token-Management-Server verteilt Tokens die Lesen oder Schreiben von Dateien oder Metadaten von Dateien erlauben. Dies ermöglicht, die Konsistenz der Dateisystem-Daten und -Metadaten zu erhalten, wenn unterschiedliche Knoten auf eine Datei zugreifen möchten. Der Token-Status wird an zwei Orten gespeichert:

- Auf dem Token-Management-Server
- Auf dem Token-Management-Klienten welcher den Token verwendet

Es folgt eine allgemeine Darstellung des Token-Managers, die später detaillierter beschrieben wird:

Beim ersten Zugriff eines Knotens auf eine Datei muss der Knoten eine Anfrage an den Dateisystem-Manager stellen um einen Lese- oder Schreibe-Token zu erhalten. Nachdem der Knoten einen Token erhalten hat, kann dieser lesend oder schreibend auf die Datei zugreifen ohne den Dateisystem-Manager zu kontaktieren, so lange kein anderer Knoten auf diese Datei lesend oder schreibend zugreifen möchte.

Der normale Ablauf für einen Token sieht folgendermaßen aus:

- Nachricht an den Token-Management-Server schicken
  Der Token-Management-Server gibt einen bewilligten Token zurück oder liefert eine Liste von Knoten, die im Konflikt mit dieser Anfrage stehen.
- Die Token-Management-Funktion des anfragenden Knotens hat die Zuständigkeit mit den Knoten die einen in Konflikt stehenden Token besitzen zu kommunizieren und sie dazu zu bewegen den Token frei zu geben. Dies veranlasst den Token-Management-Server mit allen Knoten – die Tokens mit Konflikten besitzen - zu kommunizieren. Damit ein Knoten einen Token freigeben kann muss der Dämon ihn freigeben. Das bedeutet dieser muss alle Sperren auf diesen Token aufheben. Unter Umständen muss auf noch nicht abgeschlossene E/A-Operationen gewartet werden.

3.3.4 Quota-Management

Sobald ein Dateisystem eingehängt wird, dass Quota unterstützt übernimmt der

Dateisystem-Manager automatisch die Verwaltung der Quotas. Das Quota-Management umfasst die Zuordnung von Platten-Blöcken an andere Knoten - welche in das Dateisystem schreiben – und Vergleich der Platten-Belegung mit Quota-Limits in bestimmten Intervallen. Um häufige Speicher-Anfragen zu vermeiden, werden mehr Platten-Blöcke als benötigt werden reserviert (vgl. [IBM22005], S. 66).

3.4 Logging and Recovery

In einem großen Dateisystem macht es keinen Sinn nach jedem Einhängen eines Dateisystems oder immer wenn ein Knoten heruntergefahren wird, ein Dateisystem-Check (fsck) durch zu führen. Stattdessen erfasst GPFS alle Aktualisierungen der Metadaten in einem Journal oder Write-Ahead-Log auf. Benutzer-Daten werden dabei nicht erfasst (vgl. [ScHa2002], S. 3).

Jeder Knoten besitzt ein eigenes Log in jedem Dateisystem das eingehängt wird. Da dieses Log von allen anderen Knoten gelesen werden kann, kann jeder Knoten eine Wiederherstellung eines fehlerhaften Knotens durchführen. Dadurch ist es nicht nötig auf einen fehlerhaften Knoten zu warten, bis dieser wieder einsatzbereit ist.

Nach einem Fehler ist es einfach die Systemkonsistenz wieder herzustellen. Es werden alle Updates angewendet die in dem Log des fehlerhaften Knotens gespeichert sind. Um beispielsweise eine neue Datei zu erzeugen muss der Verzeichnis-Block sowie der I-Node der neuen Datei aktualisiert werden. Nachdem der Verzeichnis-Block und der I-Node gesperrt sind, werden beiden im Puffer-Cache aktualisiert und Puffer-Logs werden erzeugt, welche beide Aktualisierung beschreibt.

Um beispielsweise eine neue Datei an zu legen muss der Verzeichnis-Block sowie der I-Node der Datei aktualisiert werden. Nachdem der Verzeichnis-Block und der I-Node gesperrt wurden, werden beide im Puffer-Cache aktualisiert ein Log-Protokolle erzeugt, welche die Aktualisierungen enthalten. Bevor der aktualisierte Verzeichnis-Block oder der I-Node wieder auf die Platte geschrieben werden können, müssen die Log-Protokolle auf die Platte geschrieben werden. Fällt z. B. der Knoten aus, nachdem der Verzeichnis-Block auf die Platte geschrieben wurde, aber bevor der I-Node geschrieben wurde, enthält die Log-Datei des Knotens das Log-Protokoll, durch welches die fehlende I-Node-Aktualisierung durchgeführt werden kann.

Sobald die Aktualisierungen durchgeführt wurden, werden die Log-Protokolle nicht mehr

benötigt und können gelöscht werden. Das bedeutet, dass die Logs eine feste Größe haben können, da Platz jederzeit durch das Anwenden der Log-Protokolle im Hintergrund freigegeben werden kann.

GPFS bewahrt die Platten-Atomität durch eine Kombination von starren Sequenzen von Operationen und Logging. Die Datenstrukturen, welche verwaltet werden sind der I-Node, der indirekte Block, die Allocation-Map und die Datenblöcke. Datenblöcke werden auf die Platte geschrieben bevor Kontrollstrukturen - die auf die Daten verweisen – auf die Platte geschrieben werden. Dieses stellt sicher, dass die vorherigen Inhalte eines Datenblocks nicht in einer neuen Datei sichtbar werden. Allokations-Blöcke, I-Nodes und indirekte Blöcke werden so auf die Platte geschrieben und Logs erstellt, dass es niemals einen Zeiger auf einen nicht allokierten Block gibt, der nicht wieder aus einem Log hergestellt werden kann. Es gibt einige Fälle, in denen Blöcke als allokiert markiert werden aber nicht Teil einer Datei sind. GPFS repliziert alle Logs. Es existieren 2 Kopien von jedem Log auf jedem Knoten (vgl. [IBM22005], S. 69).

Log-Wiederherstellung (Log-Recovery) wird in den folgenden Situationen ausgeführt:

1. Als Teil einer Wiederherstellung eines fehlerhaften Knotens, was die Objekte betrifft, die der Knoten unter Umständen gesperrt hat.
2. Als Teil eines Einhängevorgangs, nachdem das Dateisystem ausgehängt wurde.

3.5 GPFS-Dateisystem

- Die maximale Anzahl der eingehängten Dateisysteme – pro Knoten - beträgt 32.
- Die maximale Dateigröße beträgt 2^99 bytes, das entspricht 576460752303423488 TByte. Momentan sind 200 TByte getestet (vgl. [IBM32005]).
- Die maximale Anzahl der Dateien in einem Dateisystem beträgt 2.147.484.647.

3.5.1 Metadaten

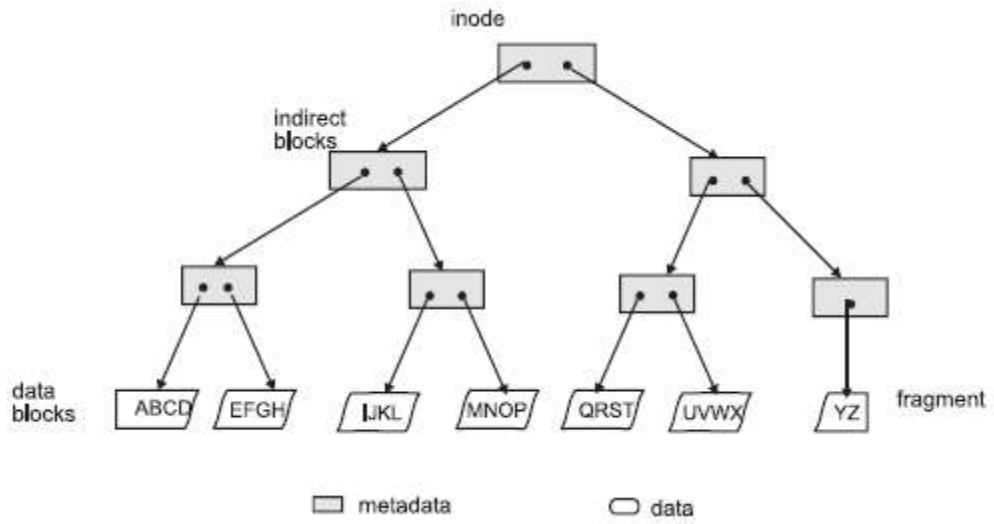

Abbildung 4: GPFS-Dateien haben eine Unix-ähnliche Struktur (vgl. [IBM2005], S. 69)

Dateien werden im GPFS wie in einem UNIX-Dateisystem verwaltet. Neben den eigentlichen Daten der Dateien werden Metadaten - die I-Nodes und indirekten Blöcke - gespeichert – falls das Dateisystem dies zulässt .

3.5.2 Metanode

Es exisiert ein Metanode pro offener Datei. Der Metanode ist für die Integrität der Metadaten verantwortlich. Normalerweise ist der Knoten, der die Datei für die längste Zeit geöffnet hat, der Metanode. Alle Knoten, die eine Datei bearbeiten, können Daten direkt lesen und schreiben aber Aktualisierungen der Metadaten werden nur von dem Metanode geschrieben. Der Metanode einer Datei ist unabhängig von den Metanodes der anderen Dateien und kann auf jeden beliebigen Knoten verlagert werden, um den Anforderungen eines Programms gerecht zu werden (vgl. [IBM22005], S. 67).

3.5.3 Block-Allocation-Map

Die Block-Allocation-Map ist eine Kollektion welche die Verfügbarkeit von Plattenspeicher auf den Platten in dem Dateisystem aufzeigt. Eine Einheit in der Allocation-Map repräsentiert einen Subblock oder 1/32 der Blockgröße des Dateisystems. Sie ist in Regionen aufgeteilt, welche sich in den Grenzen der Platten-Sektoren befinden. Die Anzahl der Regionen wird bei der Erstellung des Dateisystems durch Parameter festgelegt, welche festlegen, wie viele Knoten auf dieses Dateisystem zugreifen werden. Die Regionen werden separat gesperrt. Daraus resultiert, dass unterschiedliche Knoten aus unabhängig und gleichzeitig aus unterschiedlichen Regionen Plattenplatz reservieren oder freigeben können (vgl. [IBM22005], S. 67).

3.5.4 I-Node-Allocation-Map

Die I-Node-Allocation-Map repräsentiert die Verfügbarkeit von I-Nodes innerhalb der I-

Node-Datei. Diese Datei enthätlt alle Dateien, Verzeichnisse und Links, welche erstellt werden können. Mit Hilfe des Kommandos mmchfs kann die maximale Anzahl der Dateien festgelegt werden, die in dem Dateisystem angelegt werden können (vgl. [IBM22005], S. 68)

3.5.5 Quota-Dateien

Für Dateisysteme auf denen Quota aktiviert ist, existieren 2 Quota-Dateien:

1. user.quota
2. group.quota

Diese Dateien beinhalten die Limits für Benutzer und Gruppen. Nur der Dateisystem-Manager hat Zugriff auf diese Dateien (vgl. [IBM22005], S. 69).

3.6 Parallelität und Konsistenz

Paralleles lesen und schreiben auf ein Ziel von mehreren Knoten eines Clusters muss genau synchronisiert werden, andernfalls würden sowohl die User- als auch die Meta-Daten des Dateisystems verfälscht werden. GPFS nutzt das Distributed-Locking-Protokoll um die Konsistenz des Dateisystem unabhängig von der Anzahl gleichzeitig lesend oder schreibend zugreifender Knoten gewährleisten zu können. Gleichzeitig wird genügend Parallelität zugelassen, so dass das Maximum an Durchsatz erreicht wird.

3.6.1 Data-Striping und Allokation, Prefetch und Write Behind

Um einen hohen Durchsatz bei Dateien zu erreichen, ist es notwendig große Dateien auf mehrere Platten und mehrere Platten-Controller zu verteilen. GPFS implementiert das Striping direkt im Dateisystem und benötigt keinen separaten LVM (vgl. [ScHa2002], S. 2). Große Dateien werden in gleich große Blöcke aufgeteilt – aufeinander folgende Blöcke werden dabei in einem Round-Robin-Verfahren auf die Platten verteilt.

Das Round-Robin-Verfahren ist das simpelste Scheduling-Verfahren, bei dem angenommen wird, dass alle Prozesse die gleiche Priorität haben und diesen Prozessen folglich gleich viel Zeit zuteilt (vgl. [WiPe2005]).

Um einen Positionierungs-Overhead zu vermeiden beträgt die Block-Größe zwischen 16KByte und 1 MByte. Große Blöcke erlauben es, eine große Datenmenge mit einer E/A-Aktion zu erhalten.

GPFS speichert kleine Dateien und die Enden von großen Dateien in kleineren Einheiten – sog. Sub-Blöcke, die ein 1/32 der Größe eines normalen Blocks groß sind. Dateien, die kleiner als ein Block sind werden in Fragmenten gespeichert, die aus einem oder

mehreren Sub-Blöcken bestehen können (vgl. [IBM22005], S. 29).

Um Parallelität zu gewährleisten, wenn eine große Datei von einer Single-Thread-Applikation gelesen wird, liest – prefetched - GPFS die Daten in einen Puffer-Pool – unter Verwendung von parallelen E/A-Kommandos auf so vielen Platten wie möglich um die Bandbreite der Switching-Fabrik voll aus zu reizen.

Gleichzeitig werden die schmutzigen Daten-Puffer – auf die nicht mehr zugegriffen wird – parallel auf die Platte geschrieben. Dieser Ansatz erlaubt lesen/schreiben von/in eine einzelne Datei mit der aggregierten Datenrate, die von dem darunter liegenden Subsystem und Interconnection-Fabrik unterstützt wird. Eine Interconnection-Fabrik ist ein Geflecht aus mehreren, simultanen, dynamisch allokierbaren Transaktionen, d. h. von Knoten zu Knoten.

GPFS erkennt sequentielle, invers sequentielle und unterschiedliche Varianten von strided Patterns. Programme die nicht in diese Muster passen können ein Interface verwenden, welches Prefetch-Hints an das Dateisystem senden kann. Das Striping funktioniert am besten wenn die Platten die gleiche Größe und Geschwindigkeit aufweisen. Falls die Größen unterschiedlich sind sinkt der Durchsatz, da auf größere Platten größere Datenpakete als auf die kleinen geschrieben wird, was den Durchsatz verringert. Der Administrator kann zwischem hohen Durchsatz oder Platzmaximierung abwägen.

3.6.2 Support von großen Verzeichnissen

GPFS verwendet erweiterbares Hashing um Millionen von Dateien in einem Verzeichnis zu verwalten. Bei Verzeichnissen, die mehr als einen Block belegen wird eine Hash-Funktion auf den Namen angewendet um den korrespondierenden Block ausfindig zu machen. Dabei stehen die n niedrigen Bits des Hash-Wertes für die Block-Nummer – n hängt dabei von der Größe des Verzeichnisses ab.

Sobald das Verzeichnis anwächst, fügt das erweiterbare Hashing neue Verzeichnis-Blöcke hinzu. Falls in dem Block, der den Hash-Wert des Namens aufnimmt kein Platz mehr verfügbar ist, wird der Block in 2 Hälften geteilt. Die logische Block-Nummer des neuen Verzeichnis-Blocks errechnet sich folgendermaßen:

Hinzufügen einer 1 an der Stelle n + 1 der alten Blocknummer.
Dateien mit einer 1 an der Stelle n + 1 des Hash-Wertes werden in den neuen Block verschoben. Die anderen Verzeichnis-Blöcke bleiben von dieser Aktion unberührt.

Ein Verzeichnis ist so etwas wie eine Datei mit Löchern. Die Löcher stehen für Verzeichnis-Böcke, die noch nicht geteilt wurden. GPFS kann durch Überprüfen der Löcher in der Datei feststellen wie oft ein Verzeichnis-Block geteilt wurde und wie viele Bits eines Hash-Wertes verwendet werden müssen um den Block mit dem richtigen Namen zu finden. Das bedeutet, unabhängig von der Größe der Verzeichnis-Datei wird nur ein einziger Zugriff auf einen Verzeichnis-Block benötigt.

3.6.3 Distributed- vs. Centralized-Locking

Cluster-Datei-Systeme erlauben die Skalierung des E/A-Durchsatzes über die Leistungkapazität eines einzelnen Knotens. Um dieses Potential ausschöpfen zu können ist es notwendig, dass paralleles lesen und schreiben von allen Knoten des Clusters implementiert wird. Anderseits wird Parallelität potenziell durch den Overhead zur Erhaltung der Konsistenz des Datei-Systems und den notwendigen Synchronisierungsaufwand beim Zugriff auf Dateien bei gleichzeitiger Einhaltung von POSIX-Semantik geschmälert.

GPFS garantiert den selben Standard (POSIX) für Zugriffe auf einzelne Knoten innerhalb des gesamten Clusters, versuchen beispielsweise zwei Prozesse unterschiedlicher Knoten im Cluster auf die selbe Datei lesend zuzugreifen wird ein Knoten entweder alles oder überhaupt nichts der - vom anderen Knoten konkurrierend beschriebenen - Datei zu sehen bekommen. Hierdurch werden Lese und Schreib Operationen atomar gehalten.

Es gibt zwei verschiedene Ansätze um oben genannte Anforderungen für Synchronisation zu erreichen:

Verteiltes Locking (Distributed-Locking) bedeutet, das jede Operation auf das Dateisystem einen passenden Read- oder Write-Lock benötigt, um konkurrierende Lese oder Schreib-Zugriffe auf Daten oder Metadaten synchronisiert.

Zentrales Management (Centralized-Management). Hier werden alle konkurrierenden Operationen zu einem hierfür ausgewählten, zentralen Knoten weitergeleitet der dann die geforderten Änderungen durchführt.

Die GPFS-Architektur basiert auf dem Distributed-Locking (vgl. [ScHa2002], S. 3). Das verteilte Locking ermöglicht mehr Parallelität als der zentrale Ansatz, so lange

verschiedene Knoten auf verschiedene Teile von Daten bzw. Metadaten zugreifen. Andererseits leistet das Centralized-Management bessere Dienste sobald häufig von verschiedenen Knoten aus Zugriffe erfolgen. Dies erklärt sich dadurch das der Overhead für das Verteilte Locking das einfache Weiterleiten an den Zentral-Knoten übersteigt. Eine weitere wichtige Einflussgröße spielt hierbei die Granularität der Locks. Man spricht von kleiner Granularität wenn durch häufige Lock Anfragen Overhead entsteht. große Granularität ist gegeben, sobald häufig Lock-Konflikte entstehen.

Um effizient eine Große Bandbreite von Anwendungen zu unterstützen, werden mehrere Herangehensweisen benötigt. Zugriffs-Charakteristiken variieren je nach Auslastung und sind unterschiedlich für unterschiedliche Datentypen, wie User-Daten im Vergleich zu Meta-Daten. Folglich setzt GPFS eine Vielzahl von Methoden ein um verschiedene Daten zu verwalten. Hierzu gehört das Byte-Range-Locking speziell für Änderungen oder Updates auf die Benutzer-Daten, dynamisch ausgewählte Metanodes für das zentralisierte Management von Metadaten, sowie Mischverfahren beider Methoden für Plattenplatz-Allokationen.

Der Distributed-Lock-Manager von GPFS nutzt einen zentralen Global-Lock-Manager der auf einem der Cluster-Knoten implementiert wird und mit den jeweiligen lokalen Lock-Managern auf den restlichen Knoten zusammenarbeitet. Der Global-Lock-Manager koordiniert Locks zwischen den einzelnen Lock-Managern indem er so genannte Lock-Tokens verteilt. Diese Tokens übertragen das Recht Locks lokal zu verteilen ohne jedes Mal eine Nachricht für einen erworbenen oder freigegebenen Lock senden zu müssen. Wiederholtes Zugreifen auf den Inhalt einer Platte von demselben Knoten benötigt somit nur einmalige Erwerben des Lock-Tokens, sobald ein Knoten einen Lock-Token vom Global-Lock-Manager erhalten hat, können alle darauf folgenden Operationen - die auf dem selben Knoten ausgeführt werden - einfach das Lock-Token nutzen ohne wieder eine Nachricht verschicken zu müssen. Nur im Falle, dass ein Vorgang eines anderen Knoten einen konkurrierenden Zugriff auf das selbe Objekt benötigt, ist es notwendig über das Nachrichtensystem dem ersten Knoten den Lock zu entziehen und ihn dann dem neuen Vorgang zu gewähren.

Lock-Token spielen auch eine wichtige Rolle für die Cache-Konsistenz zwischen Knoten. Ein Token erlaubt seinem Besitzer die betroffenen Daten in seinem Cache zu halten, da diese Daten nicht von einem anderen Knoten verändert werden können.

3.6.4 Paralleler Daten-Zugriff

In einem Cluster treten häufig Anwendungen auf die das gleichzeitige Schreiben auf eine Datei von mehreren Knoten aus verlangen. GPFS verwendet Byte-Range-Locking um Lese- und Schreibzugriffe miteinander zu synchronisieren. Dieses Verfahren erlaubt es Programmen gleichzeitig schreibend auf unterschiedliche Teile einer Datei zu zu greifen. Hierbei wird der POSIX Standard für Read-/Write-Atomicity eingehalten. Würden Byte-Range-Locks so lange gehalten bis eine Anwendung fertig ist und dann den jeweiligen lock weitergeben, würde der hierdurch entstehende Locking-Overhead zu hoch werden. Daher nutzt GPFS einen fortschrittlicheren Ansatz, das Byte-Range-Locking-Protokoll, welches für die meisten Zugriffsmuster eine signifikante Reduzierung des Lock-Traffic erreicht.

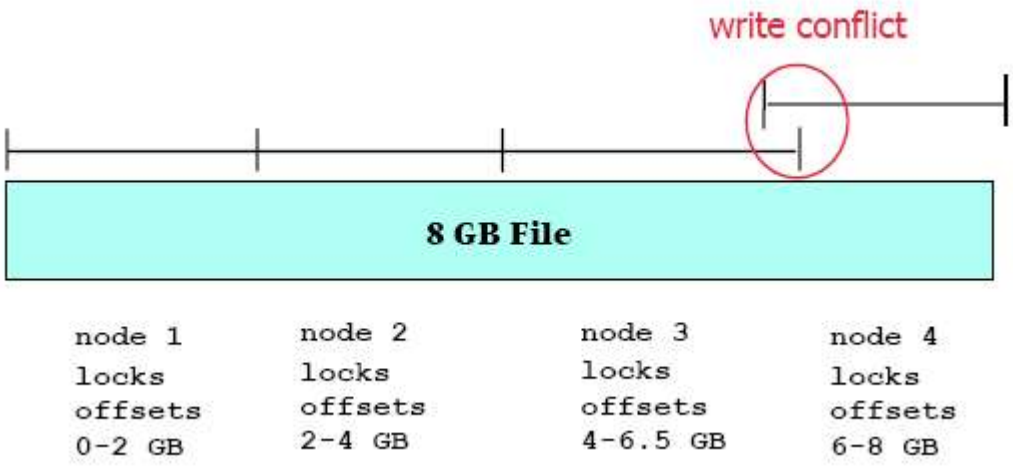

Abbildung 5: Byte-Range-Locking mit Schreibkonflikt

.

1. Knoten1 erhält einen Byte-Range-Token für die gesamte Datei – von 0 bis unendlich. Alle Lese- und Schreibzugriffe gehen lokal vonstatten, ohne Interaktionen zwischen den Knoten.
2. Knoten2 möchte in die gleiche Datei schreiben.
3. Knoten1 überprüft, ob Datei noch verwendet wird.

Falls die Datei geschlossen wurde:

4. Knoten2 übernimmt den kompletten Token

In diesem Fall verhält sich das Byte-Range-Locking wie bei einer normalen Token-Vergabe, was sehr effizient ist.

Falls die Datei von Knoten1 nicht geschlossen wurde:

4. Knoten1 gibt einen Teil des Byte-Range-Tokens frei - Knoten2 übernimmt diesen.
5. Falls Knoten1 an Offset o1 und Knoten2 an Offset o2 sequentiell schreiben, wird Knoten1 den Token von o2 bis unendlich freigeben (falls o2 > o1) oder von 0 bis o1 (falls o2 < o1).
6. Damit können beide Token ohne Token-Konflikt in die Datei schreiben.

Das Token - Protokoll wird nur Token - Anfragen abweisen, falls der angeforderte Bereich im Konflikt mit anderen Token steht (Knoten 3 und 4). Der Token Manager wird in diesem Fall wenn möglich einen Teil-Bereich freigeben falls dieser nicht mit anderen Knoten in Konflikt steht.

4 Fehlertoleranz

Aufgrund der Tatsache das Cluster-Systeme aus unabhängig und redundant ausgelegten Computern zusammengesetzt werden, ist es unwahrscheinlich das alle Komponenten zu jeder Zeit einwandfrei arbeiten. Um trotzdem eine reibungslosen und somit zuverlässigen Dienst anbieten zu können, ist es wichtig Ausfälle mit der nötigen Aufmerksamkeit zu behandeln ohne den Ablauf der laufenden Operationen zu unterbrechen zu müssen. Typische Defekte in einem Cluster können hierbei in einem Knoten, im Speicher und in der Kommunikation auftreten.

4.1 Knoten-Defekte

Wenn ein Knoten ausfällt, kümmert sich GPFS darum das Metadaten die von der ausgefallenen Komponente bearbeitet wurden wieder in einen konsistenten Zustand gebracht werden, weiterhin müssen Ressourcen wie Lock-Tokens freigegeben werden und notwendiger Ersatz für die Rolle der ausgefallenen Komponente muss beschafft werden bzw. einer anderen Komponenten übergeben werden.

Da GPFS Recovery-Logs in dem gemeinsam genutzten Speicher aufbewahrt werden, können Metadaten Inkonsistenzen - die aufgrund eines Knoten Ausfalls entstehen - ohne großen Aufwand und innerhalb kürzester Zeit durch andere nicht betroffene Knoten repariert werden. Nachdem die Wiederherstellung durchgelaufen ist, gibt der Token-Manager die vom ausgefallenen Knoten gehaltenen Tokens wieder frei. Das Distributed-Locking-Protokoll stellt sicher das ein Knoten zum Zeitpunkt seines Ausfalls wirklich alle Tokens besessen hat - sprich auch Tokens für veränderte Metadaten die sich im Cache des Knoten befanden aber noch nicht wieder in den Speicher geschrieben wurden. Da alle Tokens erst freigegeben werden nachdem der Wiederherstellungprozess fertig ist, werden auch betroffene Metadaten die durch den defekten Knoten verändert wurden, erst wieder sichtbar nachdem die Konsistenz sichergestellt worden ist. Diese Überwachung funktioniert auch für Fälle in denen GPFS im Gegensatz zum Distributed-Locking einen zentralisierten Ansatz zur Synchronisation von Metadaten-Updates verwendet. Der Trick besteht darin, dass das Update idempotent ist, d. h. nach dem Wiederherstellungsprozess kann jeder Knoten den Token des Metaknotens erhalten und stellt daraufhin fest das dieser noch keine Bestätigung für Vorgenommene Änderungen bekommen hat, hierauf schickt der Knoten einfach die Updates wieder zum Metaknoten und lässt diese wieder ausführen.

4.2 Kommunikations-Defekte

Um Knotenausfälle zu erkennen nutzt GPFS einen sog. Group-Service-Layer (vgl. [ScHa2002], S. 10) welcher durch periodische Heartbeats Nachrichten, Knoten und Kommunikationsverbindungen überwacht und ein Prozessgruppen-Zugehörigkeits-Protokoll implementiert. Wenn ein Knoten ausfällt informiert der Group-Service-Layer die verbleibenden Knoten über die Änderung in der Gruppe. Dies Löst den im vorigen Abschnitt beschriebenen Recovery- Prozess aus.

Kommunikationsfehler die durch einen schlechten Netzwerk-Adapter bzw. durch ein loses Kabel verursacht werden, bewirken das ein Knoten von den anderen Isoliert wird. Weiterhin kann ein Fehler in der Vermittlungsstelle zu einer Zerstücklung des Netzwerkes führen. Solche Aufteilungen sind nicht von einem Defekt eines Knotens zu unterscheiden, daher könnten solche partitionierte Bereiche weiterhin Zugriff auf den gemeinsamen Speicher besitzen und diesen durch diese unabhängige Arbeitsweise korrumpieren. Genau aus diesem Grund erlaubt GPFS nur der Gruppe mit der Mehrzahl von Mitgliedern im Clusterverbund den Zugriff aufs Dateisystem. Den Knoten im kleineren Verbund wird so lange der Zugriff verwehrt bis sie dem größeren Verbund wieder beigetreten sind.

Das Mitglieder-Protokoll kann keine Zusagen darüber treffen, wie lange es dauert bis ein Knoten Fehlermeldungen erhalten und verarbeitet hat.. Das bedeutet, sobald eine Partitionierung entsteht, ist es nicht klar wann die betroffenen Knoten (der kleineren Partition) informiert werden und mit dem Zugriff auf den Speicher aufhören. Bevor GPFS mit dem Recovery-Log anfangen kann, zäunt es die Bereiche die nicht mehr zum Verbund gehören ab, z. B. durch Aufruf von Funktionen des Speicher-Systems um E/A-Anfragen der betroffenen Knoten abzulehnen.

Um auch Fehlertolerante Zwei-Knoten-Konfigurationen zuzulassen, wird bei Kommunikations-Partitionierungen immer die Technik des Einzäunen (disk fencing) eingesetzt (vgl. [ScHa2002], S. 10).

4.3 Speicher Ausfälle

Da GPFS sowohl für Daten als auch Metadaten das Disk-Striping-System über alle Speicherplätze des Dateisystems nutzt, bedeutet der Ausfall einer Festplatte einen überproportionalen Einfluss auf einen großen Teil aller Dateien. Aus diesem Grund nutzen GPFS-Konfigurationen doppelt verknüpfte Raid Controller, die den physischen Ausfall

einer Platte bzw den Verlust eines Zugriffspfades abdecken können.

Als Alternative zu RAID unterstützt GPFS die Reproduktion innerhalb des Dateisystems. Wird diese Funktion aktiviert, reserviert GPFS Platz für zwei Kopien jeder Datei bzw. Metadaten-Blöcke auf je zwei unterschiedlichen Platten und speichert diese auch dorthin. Sobald eine Platte unbrauchbar wird verfolgt GPFS zurück welche replizierten Dateien sich auf dieser verloren gegangenen Platte befunden hatten. Wenn die fehlerhafte Platte wieder verfügbar ist, schaut GPFS in der anderen Kopie nach und bringt die Daten auf den neuesten Stand. Für den Fall das eine Platte nicht mehr wiederhergestellt werden kann, schafft GPFS neuen Platz für die betroffenen Blöcke auf einer anderen Platte.

5 Zusammenfassung

Die vorliegende Arbeit befasste sich vorrangig mit den Hauptfeatures des von IBM entwickelten GPFS. Für große Cluster typische Anforderungen wie hoher Durchsatz, Speicherkapazität und hohe Zuverlässigkeit werden von GPFS durch Anwendung fortschrittlicher Technologien gelöst. Vor allem zeichnet sich GPFS durch seine Fehler-Toleranz, seine Zuverlässigkeit und auch durch die vielfältigen Möglichkeiten der Skalierung aus. Im Gegensatz zu anderen Dateisystemen wie NFS oder DFS aber auch im Vergleich zu einem Parallelen Dateisystem wie PVFS erlaubt die Shared-Disk-Architektur von GPFS das jeder Compute-Node Zugriff auf alle Festplatten besitzt. Dies kann unter Verwendung des zukunftsweisenden Storage-Area-Network (SAN), z. B. Fibre Channel oder aber auch durch iSCSI realisiert werden. Der Maximal mögliche Durchsatz und ein gute Lastverteilung wird bei GPFS durch das aus dem RAID bekannten Disk – Stripping erreicht.

Durch das feinkörnige Byte-Range-Locking schafft GPFS einen anspruchsvolleren und vor allem im parallelen Bereich effizienteren Ansatz, um mit konkurrierenden Anwendungen innerhalb einer Datei performant umzugehen.

6 Abkürzungsverzeichnis

PFS	Parallel File System
GPFS	General Parallel File System
DCE	Distributed Computing Environment
SAN	Storage Area Network
FC	Fibre Channel
DFS	Distributed File System
NSD	Network Shared Disk
POSIX	Portable Operating System Interface X
SF	Switched Fabric

7 Quellen

[Hask1998] Haskin, R. L.: Tiger Shark - a scalable file system for multimedia, IBM Journal of Research and Development, Volume 42, Number 2, March 1998, pp. 185-197.

[IBM2005] IBM: GPFS overview. http://publib.boulder.ibm.com/infocenter/clresctr/topic/com.ibm.cluster.overview.doc/overview.html, 9.06.2005

[IBM22005] IBM: Concepts, Planning and Installation Guide, Version 2.3, Third edition, December 2004.

[IBM32005] IBM: GPFS V2.3 FAQs, 27.05.2005. http://publib.boulder.ibm.com/infocenter/clresctr/index.jsp?topic=/com.ibm.cluster.gpfs.doc/gpfs_faqs/gpfsclustersfaq.html, 12.06.2005.

[ScHa2002] Schmuck, F.; Haskin, Roger: GPFS: A Shared-Disk File System for Large Computing Clusters, .

[WiPe2005] Wikipedia: Round-robin scheduling, 2005. http://en.wikipedia.org/wiki/Round-robin_scheduling, 12.06.2005.

[WiPe22005] Wikipedia: Switched Fabric, 2005. http://en.wikipedia.org/wiki/Switched_fabric, 12.06.2005.

[WiPe32005] Wikipedia: Fibre Channel, 2005. http://en.wikipedia.org/wiki/Fibre_Channel, 12.06.2005.

[ZaSi2003] Zabala, S.: Das parallele Dateisystem PVFS im Vergleich mit GPFS, WS 2002/03. http://pvs.informatik.uni-heidelberg.de/Teaching/LCC-0203/zabala.pdf, 12.06.2005.