Abteilung Parallele und Verteilte Systeme
Institut für Informatik
Ruprecht-Karls-Universität Heidelberg

Bachelor-Arbeit

Änderung der Datenverteilungsfunktion im parallelen Dateisystem PVFS2

Philipp Sadleder,
philipp@sadleder.de

21. Dezember 2004

Betreuer:
Prof. Dr. Thomas Ludwig,
t.ludwig@computer.org

Ich versichere, dass ich diese Bachelor-Arbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Heidelberg, 21. Dezember 2004

_____

In the current PVFS2 design, load imbalances on the data servers could negatively affect the overall I/O performance. PVFS2 features a distribution interface, which can be used to implement various distribution functions. The distribution function concept is explained. A new Flexible Distribution Function is proposed, motivated by possible inhomogeneities in the hardware or access patterns. Flexibility depends on being able to redistribute data to match the access and load profile better.

MPI-IO provides new data access semantics beyond the very limited capabilities of POSIX I/O. This interface is intended for use in parallel high performance applications. It is shown how it can be utilized to provide parallel redistribution for PVFS2.

Additionally, some aspects of our development environment are mentioned and exemplified, including the usage of a decentralized version control system.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

In modernen Clustern stehen Festplattenkapazitäten im Terabyte- bis Petabyte-Bereich zur Verfügung[1]. Es stellt sich die Frage, wie dieser meist auf die einzelnen Knoten verteilte große verfügbare Speicherplatz effizient genutzt und verwaltet werden kann. Sollte nicht die aggregierte Schreib/Lese-Bandbreite der verwendeten Festplatten zur Verfügung stehen? Kann diese durch verteilt laufende Anwendungen angemessen ausgenutzt werden? Gibt es Möglichkeiten, durch eine andere Art der Datenverteilung den Zugriff zu optimieren?

Parallele Dateisysteme dienen zur effizienten Verwaltung großer Datenmengen für parallele Anwendungen im Bereich des Hochleistungsrechnens. Dabei können die Anwendungen auf mehreren tausend Rechnern ausgeführt werden. In [6] werden Konzepte für parallelen Datenzugriff beim Hochleistungsrechnen und Forschungsansätze für Leistungsmessung und Lastausgleich vorgestellt.

1.2 Aufgabenstellung

Die Aufgabenstellung der Arbeit besteht darin, die Verteilungsfunktion im parallelen Dateisystem PVFS2 [11] variabel zu machen und die Verteilung dynamisch zu gestalten. Verschiedene Möglichkeiten der Vorverteilung von Daten sind denkbar. Diese soll dem Zweck dienen, ein paralleles Programm, welches das *Message Passing Interface* (MPI) [7] verwendet, möglichst schnell auszuführen. Dafür lassen sich prinzipiell verschiedene Ansätze treffen. Hier wird der Weg beschritten, die Umverteilung über Bibliotheksfunktionen anzubieten, die wiederum auf MPI aufsetzen und von PVFS2 unabhängig sind. Damit kann gewährleistet werden, dass Clients weiterhin ohne Beteiligung der Datenserver entscheiden können, wo die angefragten Daten platziert sind.

Eine andere Möglichkeit stellt die Umverteilung durch Kommunikation zwischen den PVFS2-Datenservern dar. Diese wird jedoch an dieser Stelle vernachlässigt, da hierdurch weitere Synchronisationsprobleme entstünden.

Die Ansätze zur Änderung dieser Verteilungsfunktion und ihrer unterschiedlichen Parametrisierung sind Thema dieser Arbeit.

[1] zum Beispiel der "Heidelberg Linux Cluster (HELICS)" [1] siehe `http://helics.uni-hd.de/`

1.3 Abgrenzung

In der Arbeit wird ein Grundgerüst für die Möglichkeiten der Umverteilung dargestellt. Die Implementierung ist jedoch nicht so weit gediehen, dass eine automatische Umverteilung möglich wäre. Die Arbeiten erstrecken sich über mehrere Softwareprojekte, daher ist das Übersetzten und Testen sehr aufwendig und nimmt viel Zeit in Anspruch.

Die vorgestellte Möglichkeit zur Umverteilung stellt nur *einen* möglichen Ansatz dar. Dabei erfolgt die Realisierung nicht durch Interserver-Kommunikation, sondern als eigenständiges Programm durch Kopieren einer Datei. Die Kopie wird unter Verwendung einer anderen Verteilungsfunktion erstellt.

2 Das parallele Dateisystem PVFS2

PVFS2 ist ein paralleles Dateisystem, welches besonders auf die Bedürfnisse von wissenschaftlichen Anwendungen im Bereich des parallelen Hochleistungsrechnens zugeschnitten ist und unter der GPL[1] entwickelt wird.

PVFS2 ist ein paralleles virtuelles Dateisystem, welches auf bestehenden lokalen Dateisystemen aufbaut. Aus diesen formt es ein logisch zusammenhängendes Dateisystem.

PVFS2 ist vor allem auf wissenschaftliche Berechnungen ausgerichtet, welche Daten über viele Knoten verteilt gemeinsam halten und koordiniert darauf zugreifen. Es soll für die dabei typischen auftretenden Zugriffsmuster optimiert werden. Dabei sollen Daten nicht nur verteilt gespeichert werden können. Es soll auch paralleler Zugriff beim Lesen und Schreiben von vielen Clients aus möglich sein.

PVFS2 ist modular aufgebaut, um einzelne Komponenten variabel austauschen zu können. Dieser Aufbau ist aus Abb. 2.1 ersichtlich. Die Abbildung ist dem Artikel [4] entliehen, welcher einen Überblick über PVFS2 und seine Möglichkeiten bietet. Für die Netzwerkschicht von PVFS2, das *Buffered Messaging Interface* (BMI) existieren beispielsweise Implementierungen für TCP, Infiniband[2] und GM (Myrinet)[3].

Es gibt verschiedene Möglichkeiten des Zugriffs auf PVFS2. Diese sind in Abb. 2.1 schematisch dargestellt und werden folgend erläutert:

Zugriff mit nativer Schnittstelle Auf das Dateisystem kann mit der zu PVFS2 gehörenden Bibliothek `libpvfs2` zugegriffen werden. Auch die mitgelieferten Verwaltungsprogramme von PVFS2 verwenden diese Bibliothek. Einige dieser Programme werden folgend mit besonderen Optionen genannt:

`pvfs2-ls`

Der Befehl entspricht in der Funktion dem GNU `ls` Kommando. Er zeigt Informationen zu Dateien in einem PVFS2-Verzeichnis an.

`pvfs2-cp`

Der Befehl entspricht in der Funktion dem GNU `cp` Kommando. Er kopiert Dateien zwischen Linux-Dateisystemen und PVFS2-Dateisystemen. Mit der Option `-t` können Informationen über Dauer und Durchsatz des Befehls ausgegeben werden.

`pvfs2-rm`

[1] siehe GNU General Public License `http://www.gnu.org/copyleft/gpl.html`
[2] siehe Infiniband Webseite `http://www.infinibandta.org/`
[3] siehe Myrinet Webseite `http://www.myri.com/myrinet/`

Der Befehl entspricht in der Funktion dem GNU `rm` Kommando. Er entfernt Dateien aus einem PVFS2-Dateisystem.

`pvfs2-mkdir`

Der Befehl entspricht in der Funktion dem GNU `mkdir` Kommando und erstellt ein Verzeichnis in einem PVFS2-Dateisystem.

`pvfs2-chmod`

Der Befehl entspricht in der Funktion dem GNU `chmod` Kommando. Er dient zur Änderung von Zugriffsberechtigungen auf Dateien.

`pvfs2-chown`

Der Befehl entspricht in der Funktion dem GNU `chown` Kommando und ändert Besitzer- und Gruppenrechte von Dateien in einem PVFS2-Dateisystem.

`pvfs2-fsck -m <mount_point>`

Das Kommando führt eine Überprüfung des Dateisystems auf Inkonsistenzen in Daten- und Metadaten-Objekten durch. Beim derzeitigen Entwicklungsstand können dabei Fehler noch nicht automatisch behoben werden. Es können jedoch Fehler in bei den Daten und den Metadaten angezeigt und anschließend manuell beseitigt werden.

`pvfs2-mkspace`

Mit diesem Befehl kann ein *storage space*, ein für PVFS2 reservierter Speicherbereich auf einem lokalen Dateisystem, initialisiert werden. Dabei werden die benötigten Datenstrukturen für die Verwaltung der *data files* und die Datenbank für die Metadaten angelegt.

`pvfs2-ping -m <mount_point>`

Mit diesem Kommando wird überprüft, ob alle zu einer Dateisystemkonfiguration gehörenden PVFS2-Server laufen und korrekt konfiguriert sind.

`pvfs2-statfs -m <mount_point>`

Der Befehl zeigt Statistiken der einzelnen Server einer Dateisystemkonfiguration an.

Zugriff mit Linux-Kernelmodul Für PVFS2 existiert ein Kernelmodul, welches das Mounten eines PVFS2-Dateisystems ermöglicht. Bei dieser Art der Benutzung greift man mit POSIX-Semantik auf das in Linux-VFS[4] eingebundene PVFS2-Dateisystem zu. Es ist hauptsächlich als Kompatibilitätsschnittstelle gedacht. Dabei gibt es keine Möglichkeiten eines optimierten parallelen Zugriffs, da die POSIX-Semantik nicht für diese Art von E/A-Operationen vorgesehen ist. Für parallelen Datenzugriff wird eine Erweiterung der Funktionalität der E/A-Operationen nötig.

[4] Linux Virtual Filesystem Switch

Zugriff mit MPI-IO Auf das Dateisystem kann mit Hilfe der Erweiterung für Datenzugriff des *Message Passing Interface* [7] zugegriffen werden. Diese definiert verschiedene Arten komplexer Datenzugriffe, wie sie für parallele Anwendungen nötig sind. Bei dieser Art des Zugriffs können auch weitere Bibliotheken verwendet werden, welche auf einer höheren Abstraktionsebene arbeiten, wie beispielsweise Parallel-NetCDF[5] [5] oder HDF5[6].

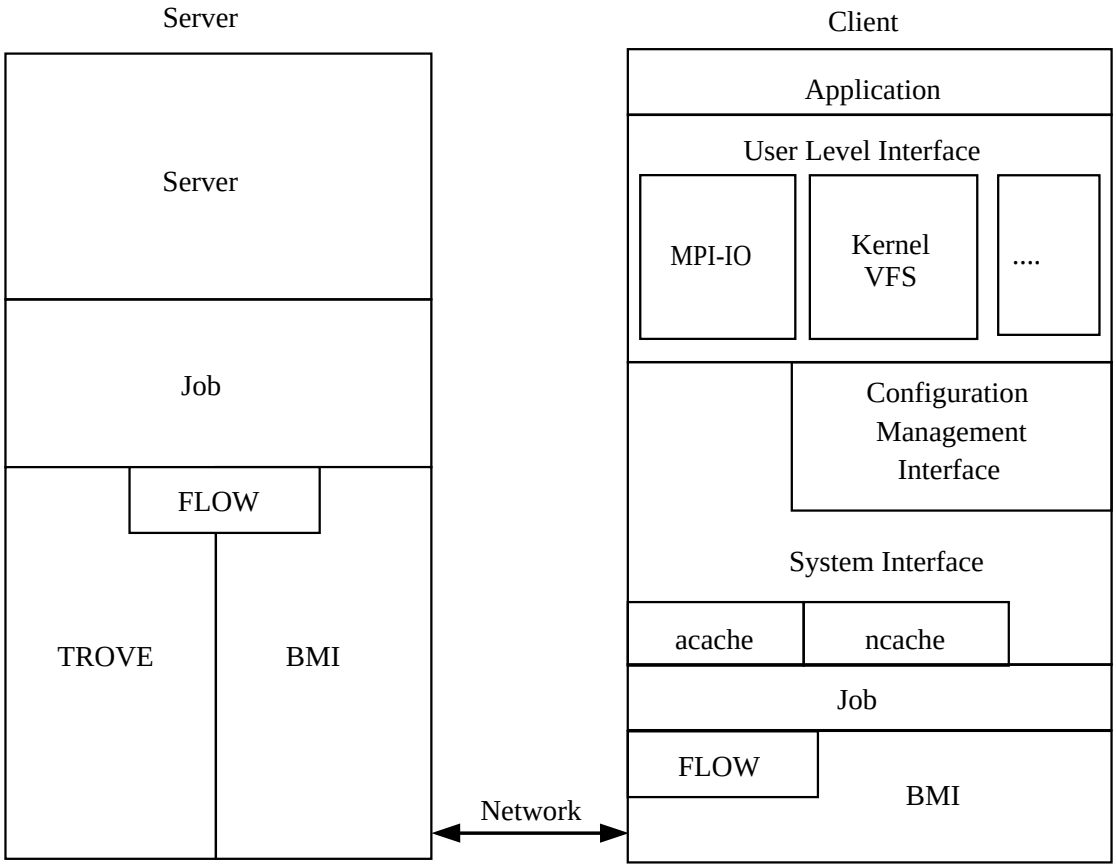

Abbildung 2.1: Aufbau von PVFS2

[5]siehe Parallel-NetCDF Webseite `http://www-unix.mcs.anl.gov/parallel-netcdf/`
[6]siehe HDF5 Webseite `http://hdf.ncsa.uiuc.edu/HDF5/`

3 Datenverteilung

3.1 Datenverteilung in PVFS2

Die Verteilung der Daten zu einer Datei geschieht in PVFS2 mittels einer Verteilungsfunktion (*distribution function*). Diese berechnet die Verteilung der Daten auf *data files*, logische Einheiten, welche einem physikalischen Datenserver zugeordnet werden. Die Datenverteilung geschieht in PVFS unabhängig von physikalischen Eigenschaften des zugrundeliegenden Netzwerks und Speichersystems.

Eine Verteilungsfunktion bietet die Möglichkeit einer von den Servern unabhängigen Berechnung der Verteilung von Daten. Ein Client kann mit der Funktion und zugehörigen Parametern, die beim Öffnen einer Datei übergeben werden, die Ziel- oder Quellposition von Daten bestimmen.

3.2 Die Schnittstelle für Verteilungsfunktionen

Innerhalb von PVFS2 gibt es Schnittstellen, die einen Austausch diverser Komponenten ermöglichen sollen. Diese werden als PVFS2-Interfaces (PINT) bezeichnet. Eine solche interne Schnittstelle gibt es auch für die Verteilungsfunktionen[1].

Eine Verteilungsfunktion wird über einen Namen angesprochen, mit dem sich auch im System registiert wird. Zugehörige Parameter werden gemeinsam mit den Metadaten zu einer Datei abgelegt und beim Zugriff auf diese wieder ausgelesen. So kann zu jeder angelegten Datei eine eigene Verteilungsfunktion und Parameter für die Verteilung angegeben werden.

Jede Verteilungsfunktion implementiert die folgenden Funktionen. Für einige der Funktionen gibt es Standard-Implementierungen[2], die meisten Funktionen jedoch müssen für jede Verteilungsfunktion einzeln implementiert werden.

```
PVFS_offset (*logical_to_physical_offset)(void* params,
                                          uint32_t server_nr,
                                          uint32_t server_ct,
                                          PVFS_offset logical_offset);
```

Diese Funktion übersetzt einen logischen Versatz in einer Datei in einen physikalischen Speicherversatz auf einem Knoten und hat diesen als Rückgabewert. Dabei wird der Knoten in `dfile_nr` als logische Einheit angegeben.

[1] Die Codebasis für die Datenverteilung ist im Quellpaket von PVFS2 zu finden in `src/io/description/`
[2] siehe `src/io/description/pint-dist-utils.c`

```
PVFS_offset (*physical_to_logical_offset)(void* params,
                                          uint32_t server_nr,
                                          uint32_t server_ct,
                                          PVFS_offset physical_offset);
```

Die Funktion übersetzt einen physikalischen Versatz auf einem Knoten in die logische Position in einer Datei.

```
PVFS_offset (*next_mapped_offset)(void* params,
                                  uint32_t server_nr,
                                  uint32_t server_ct,
                                  PVFS_offset logical_offset);
```

Die Funktion berechnet zu einem gegebenen Knoten und beliebiger Position die nächste logische Position in einer Datei, die auf diesen Knoten abgebildet wird.

```
PVFS_size (*contiguous_length)(void* params,
                               uint32_t server_nr,
                               uint32_t server_ct,
                               PVFS_offset physical_offset);
```

Rückgabewert dieser Funktion ist die Größe des zusammenhängenden Bereichs in einer Datei ab der Position `physical_offset`, der auf den Knoten abgebildet wird, welcher an die Funktion als Parameter übergeben wird.

```
PVFS_size (*logical_file_size)(void* params,
                               uint32_t server_ct,
                               PVFS_size *psizes);
```

Diese Funktion gibt die logische Größe der Datei in Byte zurück.

```
int (*get_num_dfiles)(void* params,
                      uint32_t num_servers_requested,
                      uint32_t num_dfiles_requested)
```

Die Funktion gibt die Anzahl der `dfiles`, also der zur Speicherung einer Datei verwendeten Knoten zurück. Wenn eine Datei erstellt wird, so kann die Anzahl der gewünschten Knoten übergeben werden. Diese kann höchstens der Anzahl der Knoten entsprechen, die in der Konfiguration verfügbar sind.

```
int (*set_param)(const char* dist_name,
                 void* params,
                 const char* param_name,
                 void* value);
```

Die Funktion setzt einen Parameter in der Verteilungsfunktion, wobei der Name des zu setzenden Parameters und der Verteilungsfunktion als *Strings* übergeben werden. Der Parameter wird als Zeiger vom Typ `void*` übergeben.

```
void (*encode_lebf)(char **pptr, void* params);
```

Diese Funktion wird verwendet, um die Parameter als zusammenhängenden Bereich im Speicher abzulegen. Dabei verweist der Zeiger `**pptr` auf den zu verwendenden Speicherbereich.

```
void decode_lebf(char** pptr, void* params)
```

Diese Funktion stellt die Parameter aus dem kodierten Speicherbereich wieder her.

```
void registration_init(void* params)
```

Mit dieser Funktion wird die Verteilungsfunktion in PVFS2 registriert und in die Liste der verfügbaren Verteilungsfunktionen eingetragen.

3.3 Die Standard-Verteilungsfunktion

Eine übliche Methode der Verteilung nennt man *Striping*, dabei werden die Daten blockweise im *Round-Robin*-Verfahren über die Knoten verteilt. In der Speicher- und Betriebssystemtechnik ist diese Verfahrensweise auch als *RAID 0* bekannt und wurde in [10] erstmals beschrieben. Diese Art der Verteilung wird in PVFS2 durch die Standard-Verteilungsfunktion `simple_stripe`[3] implementiert. Wird keine andere Verteilungsfunktion explizit angegeben, so wird diese verwendet.

Beim *Round-Robin*-Verfahren ist die Größe eines Datenblocks ein wichtiger Parameter, denn er wirkt sich direkt auf die Leistungsfähigkeit des Dateisystems aus. Diese Größe wird in PVFS2 als `strip_size` bezeichnet. Auch andere Faktoren spielen hierbei eine Rolle, so zum Beispiel die physikalischen und protokollbedingten Eigenschaften des verwendeten Netzwerks. Eine günstige Wahl der `strip_size` hängt von der verwendeten Hardware, der Vernetzung der Komponenten und den Zugriffseigenschaften der Anwendungen ab. Als `stripe_size` wird die Anzahl der verwendeten `data_files` mal der `strip_size` bezeichnet.

In Abb. 3.1 ist die Verteilung einer Datei mit der Verteilungsfunktion `simple_stripe` und der `strip_size=2` schematisch dargestellt.

Die Möglichkeit, ein Streifenbreite zur Laufzeit anzugeben, bestand zum Beginn der Arbeit noch nicht und musste somit erst implementiert werden. Die Vorgabe für die `strip_size` liegt bei 64 Kilobyte.

Außer der Verteilungsfunktion `simple_stripe` gibt es in PVFS2 noch die Verteilungsfunktion `basic_dist`, welche eine komplette Datei auf einen Datenserver abbildet. Sie dient hauptsächlich zu Testzwecken und soll hier nicht weiter erläutert werden.

[3] siehe `src/io/description/dist-simple-stripe.c`

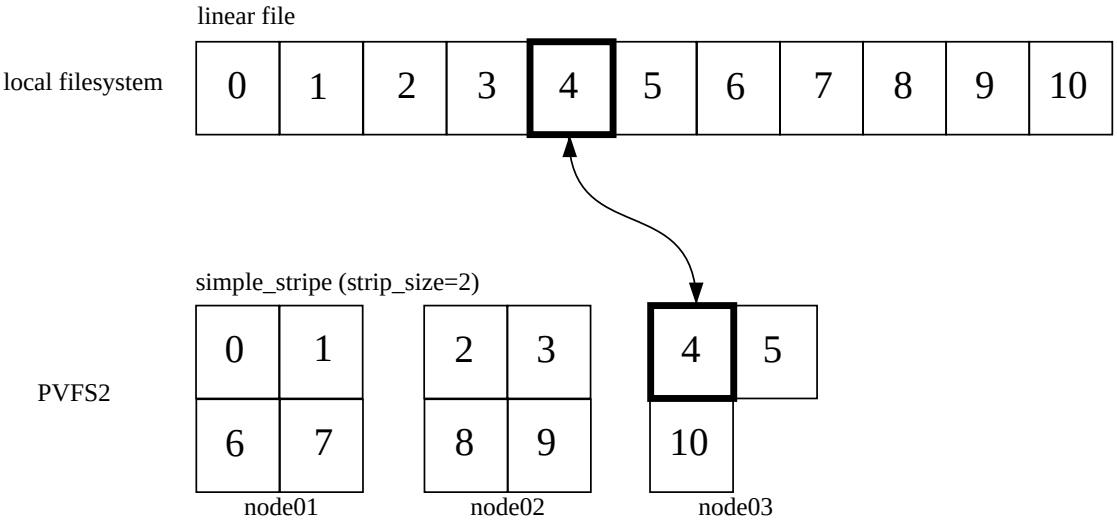

Abbildung 3.1: Standard-Datenverteilungsfunktion

4 Eine flexible Verteilungsfunktion

4.1 Beschreibung

Um herausfinden zu können, ob eine unterschiedliche Verteilung der Daten in PVFS2 mit der Leistungsfähigkeit des Datenzugriffs bei Eingabe/Ausgabe in MPI-IO in Beziehung gesetzt werden kann, sollte eine flexible Datenverteilungsfunktion implementiert werden, die eine unterschiedliche Streifenbreite pro Datenserver ermöglicht. Dies resultiert in einer ungleichen Verteilung der Daten auf die Datenserver. Bei dieser Verteilungsfunktion wird eine `strip_size` pro `data_file` angegeben.

In Abb. 4.1 ist die Verteilung einer Datei mit der Verteilungsfunktion `flexible_stripe` schematisch dargestellt.

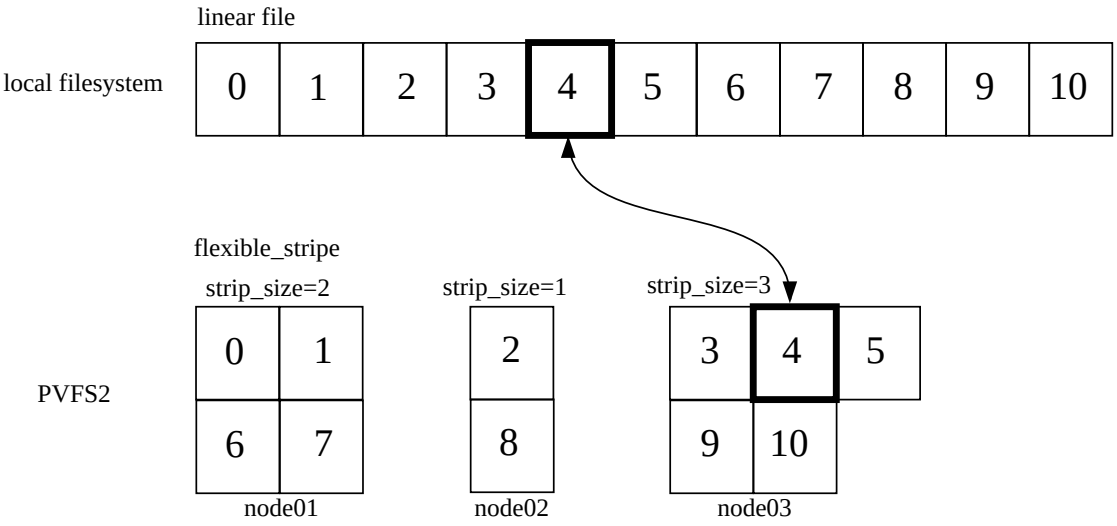

Abbildung 4.1: Flexible Datenverteilungsfunktion

4.2 Realisierung

Es wurde eine neue Verteilungsfunktion `flexible_stripe`[1] angelegt. Für jeden Datenserver kann eine unterschiedliche Streifenbreite verwendet werden. Ein Problem besteht

[1] siehe `src/io/description/dist-flexible-stripe.c`

in der Art, wie momentan Verteilungsparameter in PVFS2 verwaltet werden. Parameter werden zur Speicherung in einen fortlaufenden Speicherbereich mit Kodierungsmakros abgelegt, die nur eine Speicherung von simplen C-Datenstrukturen zulassen, nicht aber von Feldern oder anderer komplexer Datenstrukturen. Daher werden Parameter nur statisch verwendet und können nicht durch MPI-IO oder die PVFS2-Bibliothek gesetzt werden.

Da in der Verteilungsfunktion `flexible_stripe` für jedes *data file* eine eigene Blockgröße verwendet wird, enthält die Verteilungsfunktion ein *Array*, welches `strip_sizes[]` genannt wurde. In diesem wird zu jedem `data_file` die zugehörige `strip_size` abgelegt.

Parameter Versieht man Dateien in PVFS2 mit anderen als den Standard-Parametern oder wählt man eine andere Verteilungsfunktion, so möchte in der Lage sein, Informationen über diese Parameter auch wieder auslesen zu können. Grundlegende Arbeiten hierfür fehlen noch, daher wurde nur ein simpler Mechanismus zur Ausgabe dieser Parameter gewählt. Dabei wird die Ausgabe mit `printf()` direkt in den Funktionen der Verteilungsfunktion vorgenommen.

Künftig sollten die entsprechenden Werte jedoch durch die Kommandozeilenwerkzeuge von PVFS2 ausgegeben werden. Dazu müssen sie in die Datenstrukturen, in denen auch die übrigen Metadaten verwaltet werden, gespeichert und von dort wieder ausgelesen werden.

Momentan kann man in der Ausgabe von `pvfs2-ls` die Verteilungsfunktion und ihre Parameter in folgender Weise ansehen:

```
$ pvfs2-ls -lh
    drwxrwxrwx 1 sadleder sadleder    4.0K 2004-10-24 16:04 lost+found
simple_stripe@3/65536
    -r-x-w-r-x 1 sadleder sadleder 1000.0M 1970-00-02 02:33 mpiio-simple
flexible_stripe@5/{10000,1000,10000,1000,10000}
    -r-x-w-r-x 1 sadleder sadleder 1000.0M 1970-00-06 20:07 cp_out
```

Dabei wird die Verteilungsfunktion mit ihren Parametern in der Form

```
<distribution_function@<num_data_files>/<strip_size>
```

aufgeführt.

5 Umverteilung zum Lastausgleich

5.1 Lastausgleich

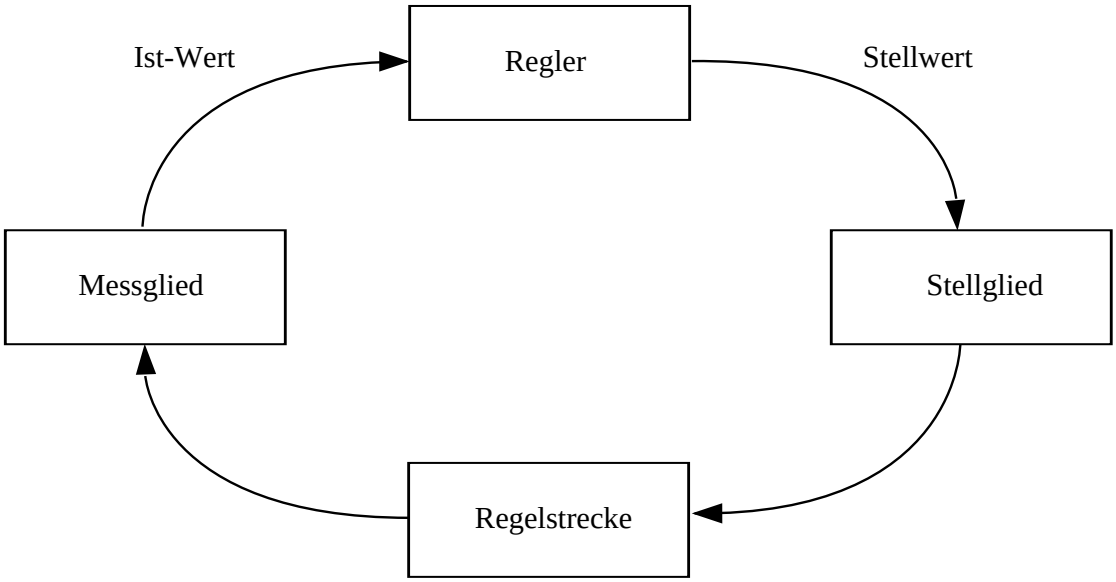

Abbildung 5.1: Regelkreis

Dem angestrebten dynamischen Lastausgleich in PVFS2 liegt die Idee eines klassischen Regelkreises[1] zugrunde. Dabei sollen im Betrieb stattfindende Messungen, wie sie in [14] beschrieben werden, zur Umverteilung von Dateien verwendet werden, um dadurch die Leistungsfähigkeit von PVFS2 ausschöpfen zu können. Folgend werden Ansätze zur Umverteilung beschrieben, die künftig in einen solchen Regelkreis, wie er in Abb. 5.1 gezeigt ist, integriert werden können.

Lastungleichheiten können einerseits strukturell durch die Verwendung heterogener Hardware-Komponenten bestehen. Dazu gehören beispielsweise Festplatten mit unterschiedlichen Leistungsmerkmalen, Netzwerkkomponenten, aber auch Prozessoren. Ande-

[1] In der Norm DIN 19226 ist der Begriff der Regelung wie folgt definiert: Das Regeln, die Regelung, ist ein Vorgang, bei dem fortlaufend eine Größe, die Regelgröße (zu regelnde Größe), erfasst, mit einer anderen Größe, der Führungsgröße, verglichen und im Sinne einer Angleichung an die Führungsgröße beeinflusst wird. Kennzeichen für das Regeln ist der geschlossene Wirkungsablauf, bei dem die Regelgröße im Wirkungsweg des Regelkreises fortlaufend sich selbst beeinflusst.

rerseits kann Lastungleichheit auch durch Ungleichgewichte in Software entstehen. Die Systemauslastung einzelner Rechnenknoten kann unterschliedlich sein, auch kann eine bestimmte Anwendung inhärent unbalanciert sein.

Im Rahmen der Arbeit sind Testprogramme entstanden, welche zwar Leistungsmessungen, jedoch noch keine automatisierte Umverteilung im Betrieb ermöglichen.

Mit der bestehenden Schnittstelle für Verteilungsfunktionen kann nicht festgelegt werden, auf welchem physikalischen Server ein `data_file` platziert wird. Es gibt jedoch bereits eine Funktion, welche die Verwendung von *data files* und die Abbildung auf physikalische Knoten des benutzten Clusters festlegt. Für eine variable Ressourcennutzung wird es nötig sein, diese Zuordnung zwischen `data_file` und E/A-Server frei bestimmen zu können, um eine Lastverteilung zu ermöglichen.

5.2 Ansätze

Für die Umverteilung in PVFS2 gibt grundsätzlich zwei verschiedene Realisierungsmöglichkeiten.

Einerseits kann man die Daten umverteilen, indem Daten direkt zwischen den Servern ausgetauscht werden. Durch das Konzept einer Verteilungsfunktion ist es nicht möglich, einzelne Blöcke von einem Datenserver auf einen anderen zu verschieben. In PVFS2 gibt es keinen *Locking*-Mechanismus für Dateien, der es erlaubte, Dateien für den Zugriff durch Clients zu sperren, bis eine solche Verschiebung stattgefunden hat.

Die andere Möglichkeit der Umverteilung stellt ein MPI-Programm dar, welches Daten aus einem PVFS2-Dateisystem ausliest und mit einer neuen Verteilungsfunktion schreibt. Dafür sind keine Änderungen an den PVFS2-Servern nötig und die Umverteilung kann innerhalb einer Anwendung durch Aufruf einer Bibliotheksfunktion ausgelöst werden.

Der zweite Ansatz wurde für diese Arbeit gewählt, da er einfacher zu realisieren ist. Außerdem kommt er mehr dem Ansatz einer Verteilungsfunktion entgegen, bei der jeder Client unabhängig von den Servern berechnen kann, wo Daten platziert werden sollen.

5.3 Realisierung

Zur Umverteilung durch Kopieren kann das in PVFS2 enthaltene Verwaltungsprogramm `pvfs2-cp` verwendet werden. Für diesen Zweck wurde das Programm in einem ersten Entwicklungsschritt so erweitert, dass eine Verteilungsfunktion und zugehörige Parameter angeben werden können. Die angegebene Datei wird kopiert und die Zieldatei unter Verwendung der angegebenen Verteilungsfunktion erzeugt. Sie kann anschließend umbenannt werden und ersetzt somit die alte Datei.

Mit der Option `-d` kann beim Kopiervorgang der Name einer Verteilungsfunktion angegeben werden, mit der Option `-s` die zu verwendende Blockgröße `strip_size`. Als Verteilungsfunktion können entweder die in PVFS2 integrierten Funktionen verwendet werden oder die hier vorgestellte Funktion `flexible_stripe`. Dabei können die Verteilungsparameter nur im Quellocode, aber noch nicht auf der Kommandozeile angegeben werden.

```
pvfs2-cp
Usage: pvfs2-cp ARGS src_file dest_file
Where ARGS is one or more of
-s <strip_size>         size of access to PVFS2 volume
-n <num_datafiles>      number of PVFS2 datafiles to use
-b <buffer_size>        how much data to read/write at once
-d <dist_name>          distribution to use for new PVFS2 files
-t                      print some timing information

-v                      print version number and exit
```

Bei dieser Art der Umverteilung wird die Leistungsfähigkeit von PVFS2 allerdings nicht ausgenutzt, da nur ein Client verwendet wird, die Umverteilung also nicht parallel erfolgt.

5.4 Parallele Umverteilung mit MPI-IO

Die verwendete Implementierung des *Message Passing Interface* ist MPICH2 [9].

MPICH2 ist eine in der Entwicklung befindliche Implementierung von MPI, welche unter einer Opensource-Lizenz[2] steht und daher frei verwendet und weiterentwickelt werden kann.

ROMIO [13] ist eine portable Implementierung von MPI-IO, dem Eingabe-/Ausgabe-Kapitel des MPI-2-Standards [8]. In MPICH2 ist ROMIO schon eingebunden, so dass es nicht als eigenständiges Packet gebaut werden muss. Eine Kernkomponente von ROMIO ist ADIO, die sogannente *abstract I/O device layer*. Für ADIO bestehen schon Implementierungen für die Anbindung einer ganzen Reihe von Dateisystemen, darunter beispielsweise auch UFS und NFS. Die Anbindung an PVFS2 befindet sich noch in der Entwicklung. Der Zugriff mit MPI-IO-Funktionen funktioniert schon, jedoch soll die ROMIO-interne Optimierung kollektiver Operationen weiter verbessert werden.

MPI Hints Im Standard des *Message Passing Interface* werden `MPI_Info`-Objekte[3] spezifiziert. Diese bestehen aus Schlüssel-Wert-Paaren, welche beide als *Strings* übergeben werden. Zu jedem Schlüssel kann nur ein Wert existieren. Manchen Funktionen in MPI-2 kann ein `MPI_Info`-Objekt als Argument übergeben werden.

In MPI-2 gibt es eine Reihe reservierter *Hints*[4]. Wenn eine Implementierung des *Message Passing Interface* einen reservierten Schlüssel benutzt, so muss er die angegebene Funktion erfüllen. Jedoch muss eine Implementierung keinen Gebrauch von den vorgegebenen Schlüsselnamen machen und kann andere Schlüssel verwenden, die nicht von MPI reserviert sind.

[2] siehe MPICH2-Lizenz `http://www-unix.mcs.anl.gov/mpi/mpich2/license.htm`

[3] siehe "The Info Object" in `http://www.mpi-forum.org/docs/mpi-20-html/node53.htm#Node53`

[4] siehe "MPI Reserved File Hints" in `http://www.mpi-forum.org/docs/mpi-20-html/node182.htm#Node183`

Werden in Anwendungen diese *Hints* verwendet, so können sie von den verwendeten Bibliotheken ausgewertet und weiterbenutzt werden.

Die Verteilungsfunktion in PVFS2 und deren Parameter sollen auch mit diesem Mechanismus angesteuert werden können. Die Benutzung von *Hints* zur Steuerung der Auswahl und Verwendung der Verteilungsfunktion war noch nicht implementiert. Da in diesem Fall ROMIO die *Hints* auswerten und eine entsprechende Ansteuerung von PVFS2 vornehmen muss, wurden auch hier Erweiterungen vogenommen. Dafür wurde die ADIO-Implementierung für PVFS2[5] so verändert, dass die folgend genannten *Hints* durch entsprechende Funktionsaufrufe der PVFS2-Bibliothek zum Einsatz kommen.

Implementierte Hints

– `striping_unit`

  Dieser *Hint* ist im MPI-2-Standard reserviert und entspricht der `strip_size` in PVFS2. Dabei wird die Blockgröße in Byte übergeben.

– `pvfs2_dist_name`

  Dieser *Hint* wird ausschließlich von PVFS2 verwendet. Der Name der zu verwendenden Verteilungsfunktion wird als *String* übergeben.

ROMIO implementiert bereits einige weitere, teilweise im MPI-2-Standard reservierte *Hints*. Folgend ist eine Ausgabe der *Hints* aufgeführt, welche bei der Öffnen einer Datei mit der Funktion `MPI_File_open()` gesetzt werden. Zur Ausgabe der *Hints* wird die Funktion `print_File_Info(MPI_Info info)` eingesetzt.

```
key = romio_pvfs2_debugmask, value = 0
key = pvfs2_dist_name, value = simple_stripe
key = striping_unit, value = 64
key = cb_buffer_size, value = 4194304
key = romio_cb_read, value = automatic
key = romio_cb_write, value = automatic
key = cb_nodes, value = 5
key = romio_no_indep_rw, value = false
key = ind_rd_buffer_size, value = 4194304
key = ind_wr_buffer_size, value = 524288
key = romio_ds_read, value = automatic
key = romio_ds_write, value = automatic
key = cb_config_list, value = *:1
```

[5] im Quellpaket von MPICH2 zu finden unter `src/mpi/romio/adio/ad_pvfs2/`

Bei diesem Funktionsaufruf wurde ein `MPI_Info`-Objekt übergeben, welches die Schlüssel `pvfs2_dist_name` und `striping_unit` enthält. Alle weiteren Schlüssel-Wert-Paare wurden von ROMIO gesetzt.

5.5 Paralleles Programm zur Umverteilung: pvfs2-mpiio-cp

Das Programm `pvfs2-mpiio-cp` wurde soweit möglich in gleicher Struktur wie der in PVFS2 enthaltene Kommandozeilen-Befehl `pvfs2-cp` gestaltet. Dafür konnte auch einiges vom schon vorhandenen Quellcode wiederverwertet werden.

Das Programm ist gegen die MPICH2-Bibliothek gelinkt und kopiert eine Datei, indem von einer beliebigen Anzahl von Prozessen jeweils Abschnitte einer Datei gelesen und mit neuer Verteilungsfunktion wieder geschrieben werden. `pvfs2-mpiio-cp` wird parallel gestartet und benutzt für jeden gestarteten Prozess einen eigenen Buffer.

Die erste Version des Programms benutzte die E/A-Funktionen `MPI_File_read_at()` und `MPI_File_write_at()`. Damit wird ein Zugriff auf das Dateisystem realisiert, wie er möglicherweise für ein normales lokales Dateisystem sinnvoll ist. Es gibt jedoch für ROMIO keine Möglichkeiten, die Anfragen zusammenzufassen und zu optimieren. In einer neuen Fassung wurden diese MPI-Funktionen durch kollektive Lese-/Schreibbefehle `MPI_File_read_all()` und `MPI_File_write_all()` ersetzt, um weitere Optimierung durch die ROMIO-Schicht zu ermöglichen.

Mit Hilfe der Funktion `MPI_File_set_view()` kann das Programm künftig unter Verwendung nicht zusammenhängender Zugriffe weiter optimiert werden. Die Art der Aufteilung der Datei wirkt sich auf die Leistung und Geschwindigkeit der Umverteilung aus.

Offen bleibt, wie die Umverteilung möglichst effektiv gestaltet werden kann. Diese Frage kann erst mit einem tieferen Verständnis der Optimierung auf den einzelnen Schichten beantwortet werden.

Ein Ausschnitt aus dem zugehörigen Makefile[6] sieht folgendermaßen aus:

```
$ mpiexec -n 10 -env PVFS2TAB_FILE ~/src/pvfs2tab \
    ./pvfs2-mpiio-cp -d simple_stripe -s $STRIPSIZE \
                     -b 20000000 -t \
                     pvfs2:/p/pvfs2-mpiio-cp_in \
                     pvfs2:/p/pvfs2-mpiio-cp_out
```

5.6 Bibliotheksfunktionen

Die Arbeit wurde begonnen mit der Absicht, einige Bibliotheksfunktionen zur Umverteilung bereitzustellen.

[6] siehe GNU Make `http://www.gnu.org/software/make/`

Dabei ist eine Funktion entstanden, welche in einem normalen MPI-Programm verwendet werden kann, um Dateien zu kopieren und sie mir einer neuen Datenverteilung zu speichern.

```
int MPIIO_copy(char *filename_in, char *filename_out,
               MPI_Offset buffer, int show_timings,
               char *dist_name, char *dist_params);
```

Die Funktion kopiert eine Datei unter Verwendung der angegebenen Verteilungsfunktion und zugehöriger Parameter. Als Parameter werden der Funktion Dateinamen der Quelldatei und der Zieldatei in der ROMIO-Notation[7] übergeben. Außerdem wird eine Puffergröße, der Name der zu verwendenden Verteilungsfunktion und Verteilungsparameter übergeben.

[7] in der Form `pvfs2://<filesystem>/<path>/<filename>`

6 Arbeitsumgebung

Neben der inhaltlichen Aufgabenstellung war auch der Aufbau einer wartbaren Entwicklungsumgebung für die Arbeitsgruppe und weitere Arbeiten im selben Gebiet Teil der Arbeit. Da bisher in der Gruppe keine Arbeiten in diesem Bereich stattgefunden haben, umfasste diese Aufgabe die Einrichtung einer Versionsverwaltung und die Erstellung von Skripten zur automatisierten Kompilieren der verwendeten Software.

Die Arbeit war durch die Verwendung zweier stark in der Entwicklung befindlicher Software-Projekte (PVFS2, MPICH2) sehr kompliziert. Einerseits wurde versucht, eine stabile eigene Entwicklungsumgebung aufzubauen, andererseits sollten aktuelle Änderungen an den verwendeten Softwareprojekten sofort genutzt werden können. Die hierfür nötige Infrastruktur war zu Beginn der Arbeit nicht vorhanden.

6.1 Versionsverwaltung: GNU Arch

Als Versionsverwaltungssystem haben wir für die Arbeitsgruppe GNU Arch [3] gewählt, da es verteilte Entwicklung, sowie auch *Branching* und *Merging* auf komfortable Weise ermöglicht. Das *Concurrent Versions System* (CVS) [2] zu verwenden kam für uns nicht in Frage, da es in diesem Versionsverwaltungssystem keine Möglichkeit gibt, Entwicklungsäste (*Branches*), welche zum gleichen Projekt gehören, in verschiedenen Archiven zu speichern. GNU Arch bietet die Möglichkeit der Entwicklung an einem Projekt, auf dessen Hauptast man keine Schreibrechte besitzt, da *Branches* in einem neuen Archiv gehalten werden können. Es gibt es bei GNU Arch keinen Grund für ein zentalisiertes Archiv, da jeder Entwickler eines Projekts seine *Branches* in einem eigenen Archiv verwalten kann.

Viele Informationen zu GNU Arch findet man im Wiki zu GNU Arch[1]. Auch gibt es dort einen Vergleich verschiedener Versionsverwaltungssysteme.

GNU Arch benötigt keine Server-Prozesse, es stützt sich nur auf die Möglichkeiten eines Dateisystems. Der Zugriff auf die Archive kann mittels verschiedener Protokollen erfolgen, darunter WebDAV, HTTP, HTTPS und SFTP. Auch die Zugriffskontrolle, also Lese- und Schreibrechte auf dem Archiv, werden gänzlich dem Betriebssystem überlassen.

Projekte im GNU Arch Archiv Im Rahmen der Arbeiten wurden die Projekte PVFS2 und MPICH2 sowie die selbstgeschriebenen Testprogramme und Skripten mit GNU Arch verwaltet. Dafür wurde ein Archiv mit dem Namen

`pvfs2-archive@ludwig9.iwr.uni-heidelberg.de--2004`

[1] siehe GNU Arch Wiki `http://wiki.gnuarch.org/`

auf dem Cluster der Arbeitsgruppe erstellt. Zu diesem Zweck wurde eine Benutzerkennung `pvfs2-archive` angelegt, mit der man sich unter Verwendung von *Public-Key*-Authentifizierung anmeldet, um auf das Archiv schreiben zu können.

Das Programm Archzoom[2] kann verwendet werden, um GNU Arch Archive in Form von Webseiten anzusehen. So kann auch das bereits erwähnte Archiv[3] durchsucht werden

Für die eigene Entwicklung wurden jeweils private *Branches* der Projekte erstellt.

Im Folgenden werden einige wichtige GNU Arch Befehle aufgeführt. Dabei werden nur wenige wichtige Optionen genannt, um einen kurzen Überblick über GNU Arch zu bieten[4].

```
tla register-archive [options] [archive] location
```

Das Archiv der Arbeitsgruppe beispielsweise kann mit dem Befehl

```
tla register-archive pvfs2-archive@ludwig9.iwr.uni-heidelberg.de--2004 \
  sftp://pvfs2-archive@ludwig9.informatik.uni-heidelberg.de/home/pvfs2-archive/2004
```

für die Benutzung registriert werden. Die Zuordnung des Namens des Archivs zum Speicherort wird in der Konfiguration von GNU Arch abgelegt und ist bei späterer Verwendung des Archivnamens bekannt.

```
tla abrowse [options] [limit]
```

Mit diese Kommando lassen sich die Inhalte eines Archivs anzeigen. Für das obengenannte Archiv erhält man folgende Ausgabe:

```
$ tla abrowse
pvfs2-archive@ludwig9.iwr.uni-heidelberg.de--2004
  mpich2
    mpich2--release
      mpich2--release--1.0
        base-0 .. patch-2

    mpich2--sadleder
      mpich2--sadleder--1.0
        base-0 .. patch-5

  pvfs2
    pvfs2--cvs
```

[2] siehe Archzoom Webseite `http://migo.sixbit.org/software/archzoom/`

[3] siehe pvfs2-archive@ludwig9.iwr.uni-heidelberg.de–2004
`http://ludwig9.informatik.uni-heidelberg.de/cgi-bin/archzoom.cgi/pvfs2-archive@ludwig9.iwr.uni-heidelberg.de--2004`

[4] Eine ausführliche Einführung in GNU Arch bietet das Tutorial
`http://regexps.srparish.net/www/tutorial/html/arch.html`

```
      pvfs2--cvs--1.0
        base-0 .. patch-678

    pvfs2--sadleder
      pvfs2--sadleder--1.0
        base-0 .. patch-95

    pvfs2--zabala
      pvfs2--zabala--1.0
        base-0 .. patch-2

  slog2sdk
    slog2sdk--release
      slog2sdk--release--1.0
        base-0 .. patch-4

    slog2sdk--zabala
      slog2sdk--zabala--1.0
        base-0 .. patch-5

  tools
    tools--mainline
      tools--mainline--1.0
        base-0 .. patch-80
```

`tla tag [options] SOURCE-REVISION TAG-VERSION`

Mit diesem Kommando wird ein *Branch* erstellt.

`tla get [options] revision [dir]`

Mit diesem Befehl wird eine Arbeitskopie eines Projekts erstellt. Er entspricht dem CVS-Befehl `cvs checkout`.

`tla update [options] [version/revision]`

Dieser Befehl bringt eine Projektkopie auf den Stand des Archivs.

`tla replay [options] [version/revision]`

Mit diesem Befehl lassen sich in einem *Branch* Änderungen von einem anderen Ast eines Projekts nachvollziehen.

Mit dem folgenden Befehl lassen sich Änderungen am *Branch* `pvfs2--cvs--1.0` in den *Branch* `pvfs2--sadleder--1.0` übertragen:

```
tla replay pvfs2--cvs--1.0
```

Diesen Vorgang nennt man *Merging*. Dabei kann es zu Konflikten kommen, die manuell beseitigt werden müssen, bevor man einen *Commit* im aktuellen Ast durchführt. Dazu betrachtet man die fehlgeschlagenen Änderungen in den `*.rej`-Dateien und führt entsprechende Änderungen an den Quelldateien durch.

```
tla commit [options] [[archive]/version] [-- file ...]
```

Mit diesem Kommando werden Änderungen an einer Projektkopie in ein Archiv eingepflegt.

```
tla changes [options] [revision] [-- limit...]
```

Änderungen an einer Arbeitskopie, die noch nicht in das Archiv eingepflegt wurden, lassen sich mit diesem Befehl anzeigen. Wird zusätzlich die Option `--diff` angegeben, so erhält man eine Ausgabe der Änderungen im Diff-Fomat.

6.2 CSCVS

Da die PVFS-Entwickler CVS [2] zur Versionsverwaltung benutzen, wird das Werkzeug `cscvs`[5] verwendet, mit dessen Hilfe zusammengehörige Änderungen (*Changesets*) aus CVS in GNU Arch importiert werden. Dieser Vorgang wird in [14] beschrieben.

Für diese Art von verteilter Entwicklung wäre es von Vorteil, wenn die PVFS2-Entwickler ein modernes Versionsverwaltungssystem verwendeten. Die Integration von Erweiterungen und Änderungen könnte beispielsweise durch einen einfachen *Merge*-Vorgang der PVFS2-Entwickler geschehen.

6.3 distcc

Das Programm `distcc`[6] ist ein Wrapper für die GNU Compiler Collection[7], der es ermöglicht, unabhängige Übersetzungsprozesse parallel auf mehreren Cluster-Knoten auszuführen. Auf den Knoten des Clusters, die am Übersetzungsprozess beteiligt sein sollen, läuft ein Serverprozess `distccd`.

Um `distcc` benutzen zu können, müssen in der Variable `DISTCC_HOSTS` die zu benutzenden Knoten angegeben werden. An `make` muss die Option `-j` übergeben werden, damit mehere unabhängige Übersetzungsprozesse nebenläufig ausgeführt werden können. Zusätzlich muss die Variable `CC=distcc` gesetzt sein, damit `distcc` als *Wrapper* für GCC eingesetzt wird.

`distcc` wurde verwendet, um PVFS2 zu kompilieren. Dadurch konnten die Übersetzungsvorgänge beschleunigt und die Entwicklung vereinfacht werden.

Für MPICH2 war dies leider nicht möglich, da im zugehörigen *Build*-System die Abhängigkeiten einzelner Quelldateien nicht korrekt modelliert sind.

[5] Changeset CVS
[6] siehe `distcc` Webseite `http://distcc.samba.org/`
[7] siehe GCC Webseite `http://gcc.gnu.org/`

6.4 Eigene Werkzeuge

Neben den anderen verwendeten Programmen haben Silvestre Zabala und ich einige Skripten und kleine Programme[8] zur einfacheren Benutzbarkeit von PVFS2 und MPICH2 geschrieben.

- `build_mpich2`

  Mit Hilfe dieses Skripts wird MPICH2 übersetzt und an einer vorgegebenen Stelle installiert. Da MPICH2 modifiziert werden musste, konnte kein systemweit installiertes Paket verwendet werden.

- `build_pvfs2`

  Unter Verwendung dieser Skripten wir PVFS2 konfiguriert und übersetzt. Dabei wird auch eine Dateisystemkonfiguration für das Cluster vorgenommen und ein *storage space* auf jedem Knoten initialisiert.

- `run_pvfs2`

  Das Skript startet die einzelnen Daten- und Metadaten-Server für PVFS2 abhängig von der jeweiligen Konfiguration.

- `build_and_run`

  Dieses Skript kombiniert die Befehle `build_pvfs2` und `run_pvfs2`.

- `rsh-pnodes`

  Mit diesem Befehl können Befehle auf allen Knoten des Clusters ausgeführt werden.

- `mpdb`

  Dieses Skript führt den Startbefehl `mpdboot` für die MPICH2-Prozessmanager mit allen in der Konfigurationsdatei `~/mpd.hosts` angegebenen Knoten aus.

6.5 Patches

Zu diversen Fehlern in PVFS2, die während der Entwicklung aufgetreten sind, haben Silvestre Zabala und ich *Patches* geliefert, die diese beheben sollen. Einige davon wurden mittlerweile in das Projekt integriert.

Bei MPICH2 konnten wir zur Beseitigung eines Fehlers beim Starten der Prozessmanager (`mpd`) beitragen.

[8] zu finden im Projekt `tools`
`http://ludwig9.informatik.uni-heidelberg.de/cgi-bin/archzoom.cgi/pvfs2-archive@ludwig9.iwr.uni-heidelberg.de--2004/tools`

7 Messungen und Testprogramme

Aufgrund der Leistungsprobleme von PVFS2 auf dem zur Entwicklung verfügbaren Cluster ist es nicht möglich, signifikante Messungen der Leistungsfähigkeit von PVFS2 durchzuführen. Das Programm zur Umverteilung `pvfs2-mpiio-cp` (siehe 5.5) sollte mit verschiedenen Verteilungsfunktionen und Parametern getestet werden.

Die Durchläufe der entwickelten Testprogramme ergeben durchweg schlecht reproduzierbare Ergebnisse. Daher beschränken sich die folgenden Abschnitte darauf, einige Erkenntnisse und Probleme zu erläutern.

7.1 Cache-Problematik

Auf der Suche nach den Problemen bei der Schreibleistung stießen Silvestre Zabala und ich wiederholt auf die Fragestellung, welchen Einfluss die auf unterschiedlichen Ebenen beteiligten *Caches* auf die Leistungsbeurteilung von PVFS2 haben. Die verwendeten Festplatten haben Hardware-Caches, das zugrundeliegende Dateisystem verwendet einen Cache im Linux-Kernel, die ROMIO-Schicht fasst einzelne Anfragen zusammen und PVFS2 selbst arbeitet wiederum mit eigenen Caches.

7.2 Konfiguration der Testumgebung

Getestet wurde auf dem Cluster der Arbeitsgruppe "Parallele und Verteilte Systeme"[1]. Eine Übersichts der Topologie des Clusters gibt Abb. 7.1. Das Cluster besteht aus 10 Knoten, wobei für die Programmläufe 5 Knoten für den Betrieb von PVFS2 eingesetzt wurden. Die verbleibenden 5 Knoten dienten als Rechnenknoten. Auf diesen wurden die MPI-Programme gestartet.

Wie sinnvoll Verteilungen mit unterschiedlicher Streifenbreite auf den einzelnen Knoten in der Praxis sein mögen, darüber kann an dieser Stelle nichts ausgesagt werden. Je nach Zugriffsschema der parallelen Anwendung könnte sich daraus möglicherweise eine bessere Auslastung des Speichersystems ergeben.

7.3 b_eff_io

Der Benchmark *parallel effective I/O bandwidth benchmark* (kurz: b_{eff_io}) [12] von Rolf Rabenseifner[2] ist ein synthetischer Benchmark, welcher typische Zugriffsmuster paralle-

[1] siehe Webseite des PVS-Clusters `http://ludwig9.informatik.uni-heidelberg.de`

[2] erhältlich unter `http://wwyw.hlrs.de/organization/par/services/models/mpi/b_eff_io/`

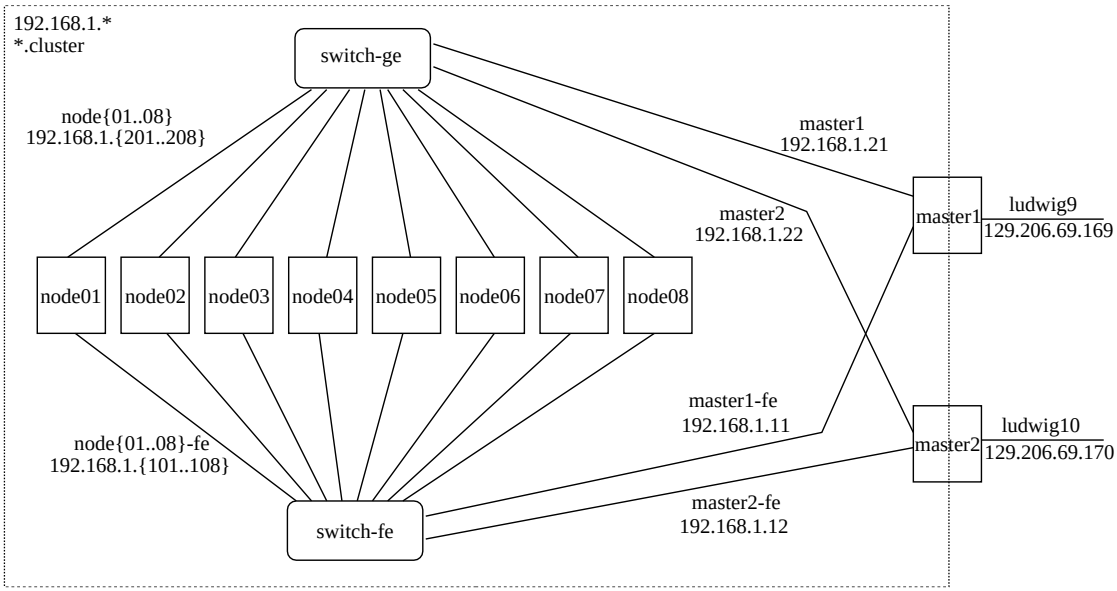

Abbildung 7.1: Cluster der Arbeitsgruppe “Parallele und Verteilte Systeme”

ler MPI-IO-Programme simulieren soll, um damit parallele Dateisysteme auf ihre Leistungsfähigkeit hin zu testen. Der Benchmark wurde von der TFCC[3] zur Aufnahme in die Bewertung der Top500[4] vorgeschlagen.

`b_eff_io` sollte verwendet werden, um Messungen mit verschiedenen Datenverteilungen durchzuführen, was aber angesichts der Leistungsprobleme, welche auf dem zur Verfügung stehenden Cluster immer noch fortdauern, keine verwertbaren ergibt.

Mit der Option `-i <file>` kann man `b_eff_io` mit einer sogenannten Info-Datei starten. Darin werden Schlüssel-Wert-Paare angegeben, welche in ein `MPI_Info`-Objekt umgewandelt und beim Start des Programms gesetzt werden. Eine einfache Info-Datei für `b_eff_io` kann beispielsweise folgenden Inhalt haben:

```
# a default hint configuration
# proc_num access pat_type chunk_size info_key        info_value

  all      all    all      all        pvfs2_dist_name simple_stripe
  all      all    all      all        striping_unit   65536
```

Bei `b_eff_io` gibt es die Möglichkeit, *Hints* nur für einzelne Ablaufsschritte des *Benchmarks* zu setzen. Zu diesem Zweck dienen die ersten vier Spalten der Info-Datei. In der dargestellten Konfiguration werden zwei *Hints* für alle Abschnitte von `b_eff_io` ge-

[3] IEEE Computer Society Task Force on Cluster Computing `http://www.ieeetfcc.org/`
[4] TOP500 SUPERCOMPUTER SITES `http://www.top500.org/`

setzt. Als Verteilungsfunktion wird dabei `simple_stripe` verwendet, die Blockgröße für die Verteilung wird auf 64 Kilobyte festgelegt.

7.4 mpiio-simple

`mpiio-simple` ist ein einfaches MPI-Testprogramm, welches einen Speicherbereich von jedem beteiligten Knoten in ein PVFS2-Dateisystem schreibt und anschließend wieder ausliest. Dafür werden die Funktionen `MPI_File_read_at()` und `MPI_File_write_at()` verwendet.

Eine Ausgabe von `mpiio-simple` sieht folgendermaßen aus:

```
$ mpiexec -n 10 -env PVFS2TAB_FILE /home/sadleder/src/pvfs2tab \
  ./mpiio-simple pvfs2:/p/mpiio-simple 100 6

Each process will read/write 100 MiB (26214400 ints)
Aggregated write bandwidth 16.915884 MiB/s
  (local: max: 1.704502, min: 1.691588)
Aggregated read bandwidth 493.056796 MiB/s
  (local: max: 17146.903234, min: 49.305680)
Aggregated write bandwidth 16.879607 MiB/s
  (local: max: 1.705993, min: 1.687961)
Aggregated read bandwidth 473.373486 MiB/s
  (local: max: 17158.126406, min: 47.337349)
Aggregated write bandwidth 16.951787 MiB/s
  (local: max: 1.708599, min: 1.695179)
Aggregated read bandwidth 477.918006 MiB/s
  (local: max: 15042.513359, min: 47.791801)
Enabling the synchronization of the clocks...
Writing logfile....
Finished writing logfile.
```

In der Ausgabe des Programms kann man sofort erkennen, das auf dem Cluster ein Problem bei der Schreibleistung von PVFS2 besteht. Diesem Problem wurde versucht nachzugehen, auch mit Hilfe der PVFS2-Entwickler, jedoch konnten bisher keine Fortschritte erzielt werden.

8 Ausblick und Fortführung der Arbeiten

Ausblick Weitere, sehr viel komplexere Verteilungsfunktionen sind möglich, welche beispielsweise mehrdimensionale Datenstrukturen von Anwendungen im Dateisystem abbilden, um eine Leistungsoptimierung vorzunehmen. Da jedoch momentan keine realen Anwendungen mit parallelen Datenzugriffen unter Verwendung von MPI-IO existieren, ergibt es wenig Sinn, solche Funktionen zu implementieren.

Weiterführende Arbeiten Als nächster Schritt sollen die geleisteten Implementierungsarbeiten in PVFS2 integriert werden.

MPI-IO-Aufrufe, die daraus resultierenden Abläufe in PVFS2 und Verteilung der Daten müssen in Beziehung zueinander gesetzt werden können. Grundlegende Arbeit hierfür wurde von Silvestre Zabala in [14] geleistet.

Sobald die Leistungsprobleme von PVFS2 auf dem Cluster gelöst sind, können Messungen mit unterschiedlichen Datenverteilungen vorgenommen werden. Die Entwicklung weiterer Datenverteilungsfunktionen, welche die in Bibliotheken Parallel-NetCDF und HDF5 verwendeten Datenstrukturen so auf das Dateisystem abbilden, dass ein optimaler Zugriff möglich wird.

Mit den PVFS2-Entwicklern soll geklärt werden, wie Parameter am besten an die Verteilungsfunktion übergeben werden sollen. Zum jetzigen Zeitpunkt geschieht dies per `void`-Zeiger auf den Speicherbereich, in dem der Parameter abgelegt, ich würde jedoch eine ähnliche Verfahrensweise wie bei den *Hints* in MPI vorschlagen.

Literaturverzeichnis

In diesem Verzeichnis und in den Fußnoten werden Verweise auf Webseiten angegeben. Diese wurden zum Abgabedatum der Arbeit auf Aktualität überprüft.

[1] Bastian, Peter und Thomas Ludwig: *Helics — ein Rechner der Superklasse*[1]. Ruperto Carola — Forschungsmagazin der Universität Heidelberg, (3):4–7, 2004.

[2] *Concurrent Versions System Webseite*. http://www.gnu.org/software/cvs/.

[3] *GNU Arch Webseite*. http://www.gnu.org/software/gnu-arch/.

[4] Kunkel, Julian, Thomas Ludwig und Hipolito Vasquez: *Weit verteilt — Dateisystem für parallele Systeme: PVFS2*. iX — Magazin für professionelle Informationstechnik, (6):110–113, Juni 2004.

[5] Li, J., W. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp und R. Latham: *Parallel netCDF: A scientific highperformance I/O interface*, 2003.

[6] Ludwig, Thomas: *Research Trends in High Performance Parallel Input/Output for Cluster Environments*. In: *Proceedings of the 4th International Scientific and Practical Conference on Programming UkrPROG'2004*, Seiten 274–281, Kiev, Ukraine, 6 2004.

[7] *Message Passing Interface Forum Webseite*. http://www.mpi-forum.org/.

[8] *MPI-2 Webseite*. http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html.

[9] *MPICH2 Webseite*. http://www-unix.mcs.anl.gov/mpi/mpich2/.

[10] Patterson, David A., G. Gibson und R. H. Katz: *A Case for Redundant Arrays of Inexpensive Disks (RAID)*. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1988.

[11] *PVFS2 Webseite*. http://www.pvfs.org/pvfs2/.

[12] Rabenseifner, Rolf, Alice E. Koniges, Jean-Pierre Prost und Richard Hedges: *The Parallel Effective I/O Bandwidth Benchmark: b_eff_io*. in Christophe Cerin and Hai Jin (Eds.), Parallel I/O for Cluster Computing, Chap. 4. (pp 107-132), Kogan Page Ltd., Feb. 2004, ISBN 1-903996-50-3, 2004.

[1] der Artikel ist auch abrufbar unter http://www.uni-heidelberg.de/presse/ruca/ruca04-03/s04heli.html

[13] *ROMIO Webseite*. http://www-unix.mcs.anl.gov/romio/.

[14] Zabala, Silvestre: *Leistungsmessung verschiedener Datenverteilungsfunktionen in dem parallelen Dateisystem PVFS2 – Bachelor Thesis*, Dezember 2004.