# Clusterfile

Johannes Müller und Florian Quadt

25. Juni 2005

Clusterfile ist ein paralleles Dateisystem, das für den Betrieb in Rechnerclustern ausgelegt ist. Es verwendet ein Partitionierungsmodell mit einer Datenrepräsentation, welche beliebige physikalische und logische Partitionierungen erlaubt. Clusterfile ist außerdem in der Lage, Konvertierungen zwischen verschiedenen Partitionierungen durchzuführen.

## Inhaltsverzeichnis

# 1 Einleitung

Rechnercluster werden vielfach für Anwendungen eingesetzt, bei denen große Datenmengen verarbeitet werden müssen. Dabei stellt sich weniger die Rechenleistung der heutigen Prozessoren, sondern viel mehr die Leistungsfähigkeit des Ein- und Ausgabesystems als Engpass heraus. Deshalb hat die Optimierung des Zugriffs auf Daten hohe Priorität.

Die meisten parallelen Dateisysteme teilen die Daten auf mehrere Rechner, sog. *I/O-Knoten* (*I/O-Nodes*), auf (siehe Abb. 1). Wenn *Rechenknoten* (*Compute Nodes*) auf eine Datei parallel zugreifen, werden diese Anfragen ggf. an unterschiedliche I/O-Knoten weitergeleitet. Diese Architektur macht das Ein-/Ausgabesystem skalierbar, indem bei gestiegenen Anforderungen weitere I/O-Knoten hinzugefügt werden. Die Art der Verteilung einer Datei auf die I/O-Knoten (*physikalische Partitionierung*) muss unbedingt an der Art der Zugriffe seitens der Rechenknoten ausgerichtet sein. Ein falsches Layout der Aufteilung kann die Geschwindigkeit und Skalierbarkeit auf verschiedene Weisen negativ beeinflussen:

- Die Daten werden stark fragmentiert auf den I/O-Knoten abgelegt. Bei Zugriffen sind komplexe und zeitaufwändige Indexberechnungen notwendig.
- Durch die Fragmentierung der Daten werden viele kleine anstatt wenige große Pakete über das Netzwerk geschickt.
- Konkurrierende Zugriffe mehrerer Rechenknoten können zu Überlastung führen und somit die Parallelität beeinträchtigen.
- Bei schlechter räumlicher Lokalität entstehen nicht-sequentielle und damit zeitaufwändige Zugriffe.

Wurde ein bestimmtes Datenlayout für die I/O-Knoten einmal festgelegt, so ist ein Umstrukturieren in ein anderes Layout unter Umständen sehr schwierig und zeitaufwändig.

Neben der physikalischen Partitionierung spielt auch die *logische Partitionierung* eine wichtige Rolle. Sie bestimmt, wie die Dateien von den Anwendungen auf den Rechenknoten gesehen werden. Abbildung 1 stellt den Zusammenhang zwischen physikalischer und logischer Partitionierung dar.

Bei Clusterfile handelt es sich um ein hochflexibles Dateisystem, das Anpassungen der physikalischen und logischen Partitionierung in hohem Maße zulässt. Es wurde erstmals 2001 auf der *IEEE International Conference on Cluster Computing* in Newport Beach, Kalifornien, von Florin Isaila und Walter F. Tichy vorgestellt. Isaila und Tichy sind beide Angehörige der Universität Karlsruhe. Zur Zeit findet man dieses Dateisystem nur im akademisch-wissenschaftlichen Umfeld.

Im folgenden Abschnitt 2 werden der Ansatz von Clusterfile motiviert und die Designziele erläutert. In Abschnitt 3 wird die zur Datenrepräsentation genutzte Struktur erklärt, die in Abschnitt 4 auf Dateien angewendet wird. Abschnitt 5 beinhaltet die Techniken, um vorhandene Datenlayouts in neue Layouts umzuwandeln. In Abschnitt 6 werden die Komponenten von Clusterfile im Detail erklärt. Abschließend folgt eine Zusammenfassung und ein Ausblick.

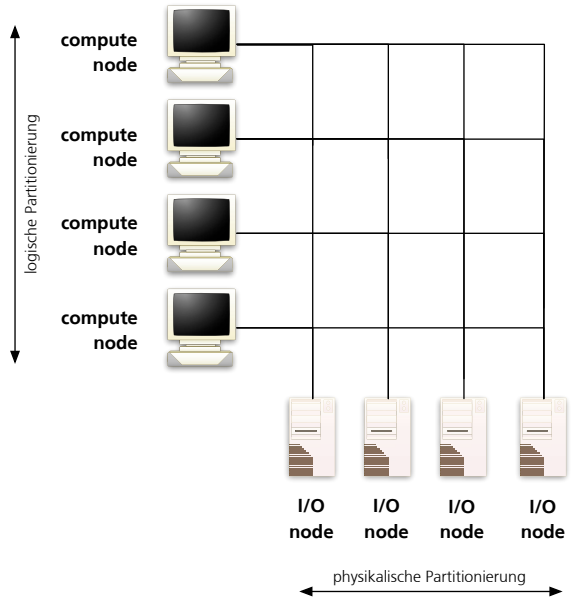


**Abbildung 1:** I/O-Knoten und Rechenknoten in parallelen Umgebungen. *Quelle:* [IT2003], S. 2.

## 2 Motivation und Designziele

Es wurden schon zahlreiche Studien im Bereich der parallelen Ein-/Ausgabe durchgeführt. Die für die Entwicklung von Clusterfile wesentlichen Ergebnisse sollen anhand eines einfachen Beispieles veranschaulicht werden.

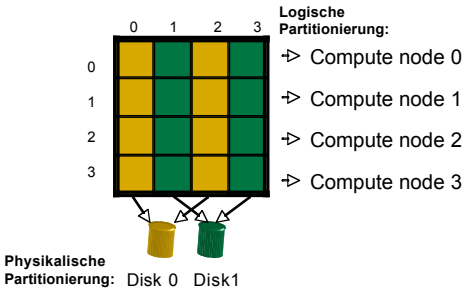


**Abbildung 2:** Spaltenweise Aufteilung einer $4 \times 4$-Matrix auf 2 Disks

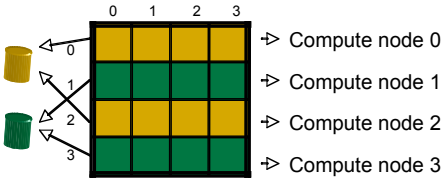


**Abbildung 3:** Zeilenweise Aufteilung einer $4 \times 4$-Matrix auf 2 Disks

Abbildungen 2 und 3 zeigen zwei verschiedene Möglichkeiten, Daten einer $4 \times 4$-Matrix physikalisch auf I/O-Server bzw. Festplatten aufzuteilen. In dem auf Abbildung 2 gezeigten Szenario wurden die Daten spaltenweise abwechselnd auf Disk 0 und Disk 1 geschrieben, während in Abbildung 3 die Aufteilung zeilenweise erfolgte. Die logische

Partitionierung ist so festgelegt, dass die Berechnung durch die Rechenknoten 0 bis 3 in beiden Fällen zeilenweise erfolgt. Wenn nun eine Matrix-Multiplikation durchgeführt wird, benötigt jeder Rechenknoten für seine Berechnung unterschiedliche Daten, d. h. die Daten sind von keinem anderen Rechenknoten im Zugriff. Deshalb kann es zu keinen Zugriffskonflikten kommen.

In vielen Fällen, kann auf die Daten nicht zusammenhängend und damit performant zugegriffen werden. Beispielsweise benötigt der Rechenknoten 0 für seine Berechnung die Matrix-Elemente $(0, 0)$ bis $(0, 3)$. Im Szenario auf Abbildung 2 befinden sich die Elemente $(0, 0)$ und $(0, 2)$ auf Disk 0 sowie $(0, 1)$ und $(0, 3)$ auf Disk 1. Die Daten können von Disk 0 und Disk 1 also nicht zusammenhängend ausgelesen werden, sondern es sind Sprünge auf der Festplatte notwendig, die Zeit kosten. In dem anderen Szenario auf Abbildung 3 besteht dieses Manko nicht und der Zugriff auf die Daten kann zusammenhängend erfolgen. Hier ist außerdem festzustellen, dass die Anforderung der notwendigen Datenpakete auf Abbildung 2 durch vier Anfragen erfolgen muss (abwechselnd Disk 0 und 1), während auf Abbildung 3 nur eine Anfrage notwendig ist.

Ein weiterer Nachteil wird durch die Art des Zugriffs auf Abbildung 2 erkennbar. Wenn die Rechenknoten abwechselnd Daten lesen, berechnen und Daten schreiben, werden die entsprechenden Schritte synchron durchgeführt. D. h. alle Rechenknoten lesen gleichzeitig das erste Element ihrer Zeile von Disk 0, berechnen den neuen Wert und schreiben wiederum gleichzeitig auf Disk 0 zurück. Da bei Disk 0 alle Anfragen gleichzeitig eintreffen (idealisiert), werden die Anfragen dort sequentiell abgearbeitet und die Vorteile der Parallelität brechen ein.

Aus diesen und weiteren Erkenntnissen (siehe [IT2003]) haben die Entwickler von Clusterfile mehrere Designziele definiert, die bei der Konzeption berücksichtigt wurden und die für die optimale Abstimmung des physikalischen Layouts an die Zugriffsmuster notwendig sind:

- Die Repräsentation für das physikalische und logische Layout des Dateisystems soll einheitlich sein und größtmögliche Flexibilität bieten. (siehe Abschnitt 3)
- Das auf der Datenrepräsentation aufbauende Partitionierungsschema soll die Einrichtung beliebiger Partitionen ermöglichen.
- Für Anwendungen, die multidimensionale Felder verwenden, soll eine kompakte Repräsentation verwendet werden.
- Es soll möglich sein, zwischen verschiedenen Verteilungsmustern effizient zu migrieren. (siehe Abschnitt 5)
- Falls ein zusammenhängender Zugriff auf Daten nicht möglich ist, soll der Zugriff durch Mapping-Funktionen und Algorithmen zur Datenumverteilung dennoch möglichst effizient erfolgen.

# 3 Datenrepräsentation

## 3.1 Nested PITFALLS

Für die Repräsentation des physikalischen und logischen Layouts verwendet Clusterfile eine Erweiterung der so genannten $PITFALLS$ (*Processor Indexed Tagged FAmily of Line Segments*). $PITFALLS$ sind kompakte Beschreibung von regulären Verteilungen. Ausgehend von der simpelsten Form *line seqment* wird im Folgenden das Konzept von *nested* $PITFALLS$ schrittweise erklärt.

**Line seqment** Ein *line seqment (LS)* ist ein Nummernpaar $(l, r)$, das einen zusammenhängenden Bereich einer Datei beschreibt. Dabei ist $l$ die linke und $r$ die rechte Begrenzung.

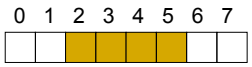


**Abbildung 4:** Die Darstellung des line segments $(2, 5)$.

**FAmily of Line Segments (FALLS)** Eine *family of line segments (FALLS)* ist ein Tupel $(l, r, s, n)$, das eine Menge von $n$ gleich großen line segments beschreibt, die untereinander den gleichen Abstand $s$ (*stride*) haben. $l$ und $r$ beschreiben den linken bzw. rechten Begrenzer des ersten line segments. Der Abstand $s$ zum vorangehenden line segment wird ausgehend vom linken Begrenzer $l$ des Vorgängers gemessen.

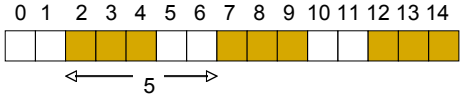


**Abbildung 5:** Die Darstellung des FALLS $(2, 4, 5, 3)$.

**Nested FALLS** Ein *nested FALLS* wird durch das Tupel $(l, r, s, n, S)$ dargestellt. Dabei ist $(l, r, s, n)$ ein äußerer FALLS und $S$ eine Menge von inneren FALLS. Zunächst wird das äußere FALLS konstruiert, dann folgt jeweils innerhalb dieser FALLS die Konstruktion der inneren FALLS. Die folgende Abbildung verdeutlicht den Aufbau eines nested FALLS.

**PITFALLS** Bei den PITFALLS wird den FALLS eine weitere Dimension hinzugefügt. Notiert werden PITFALLS als das Tupel $(l, r, s, n, d, p)$. Neu sind hier also die Parameter $d$ und $p$. $d$ ist der Abstand zweier „untereinanderliegender“ FALLS und $p$ ist die Anzahl der FALLS in dieser Dimension.

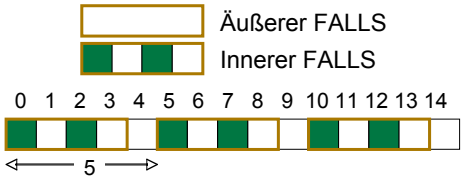


**Abbildung 6:** Die Darstellung des nested FALLS $(0, 3, 5, 3, \{0, 0, 2, 2, \emptyset\})$.

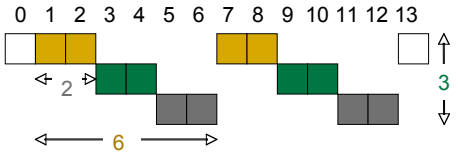


**Abbildung 7:** Die Darstellung des PITFALLS $(1, 2, 6, 2, 2, 3)$.

**Nested PITFALLS** Wie schon bei FALLS und nested FALLS handelt es sich bei Nested PITFALLS um eine Verschachtelung von PITFALLS. Ein PITFALLS wurde durch das Tupel $(l, r, s, n, d, p)$ repräsentiert. Ein nested PITFALLS besitzt eine weitere Komponente $S$, welche eine Menge weiterer PITFALLS ist: $(l, r, s, n, d, p, S)$. $(l, r, s, n, d, p)$ ist dabei die Darstellung des äußeren PITFALLS, $S$ ist ein innerer PITFALLS. Wie bei der Konstruktion von nested FALLS empfiehlt sich auch hier die Vorgehensweise, zunächst den äußeren PITFALLS darzustellen und anschließend die inneren PITFALLS einzusetzen. Auch hier soll eine Abbildung das Konzept verdeutlichen.

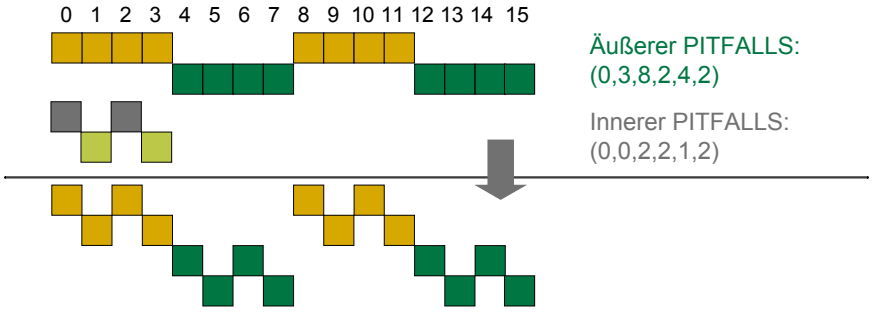


**Abbildung 8:** Die Darstellung des nested PITFALLS $(0, 3, 8, 2, 4, 2, \{0, 0, 2, 2, 1, 2, \emptyset\})$.

## 3.2 Motivation für den Einsatz von nested PITFALLS

Clusterfile verwendet Mengen von nested PITFALLS für die folgenden drei wesentlichen Anwendungen:

1. Repräsentation der physikalischen Partitionierung

2. Repräsentation der logischen Partitionierung

3. Erstellung von Mapping-Funktionen zwischen beliebigen Partitionen

Bei der Definition von physikalischen und logischen Partitionierungen soll allerdings die Komplexität der nested PITFALLS durch eine entsprechende Zwischenschicht dem Benutzer verborgen bleiben. Die Entwickler von Clusterfile begründen die Wahl von nested PITFALLS zur Datendarstellung wie folgt ([IT2003]:

**Beliebige Partitionen** Mit PITFALLS können beliebige Verteilungen dargestellt werden. Selbst Verteilungen, die keinem regulären Muster folgen, können durch PITFALLS ausgedrückt werden, indem eine Menge von line segments (setze $n = 1$ und $p = 1$) verwendet wird.

**Kompakte Darstellung von regulären Verteilungen** Komplexe reguläre Verteilungen, wie sie z. B. für die Aufteilung multidimensionaler Felder auf mehrere I/O-Knoten verwendet werden, lassen sich durch einen kurzen Ausdruck darstellen.

**Effizientes Mapping und Umverteilung** Für die Konvertierung zwischen regulären Verteilungen existieren effiziente Algorithmen für in PITFALLS-Notation dargestellte Daten (siehe Abschnitt 5).

# 4 Physikalische und logische Partitionierung

Clusterfile benutzt einheitlich PITFALLS für die physikalische und logische Partitionierung einer Datei. Die Elemente einer physikalischen Partition werden im Folgenden *Subfiles*, die Elemente der logischen Partition *Views* genannt.

## 4.1 Aufteilen einer Datei

Eine Datei (*file*) ist im Clusterfile-Modell eine linear adressierbare Folge von Bytes. Sie besteht aus einem Versatz (*displacement*) und einem Partitionierungsmuster (*partitioning pattern*). Der Versatz ist ein Offset, der eine Position relativ zum Dateianfang bestimmt. Das Partitionierungsmuster $\mathcal{P}$ besteht aus der Vereinigung von $n$ Mengen von nested FALLS $S_0, S_1, \ldots, S_{n-1}$, wobei jede ein linear adressierbares Partitionselement darstellt:

$$\mathcal{P} = \bigcup_{i=0}^{n-1} S_i$$

Die Mengen $S_i$ müssen zwei wesentliche Bedingungen erfüllen:

1. Alle $S_i$ beschreiben nicht überlappende Regionen der Datei. Es befindet sich also ein Byte der Datei auf *höchstens einem* Partitionselement.

2. $\mathcal{P}$ beschreibt die Datei zusammenhängend – also lückenlos. Jedes Byte der Datei muss also auf *mindestens einem* Partitionselement abgebildet sein.

Durch die beiden oben genannten Bedingungen ist sichergestellt, dass jedes Byte der Datei in *genau einem* Partitionselement dargestellt wird – oder genauer: Das Byte wird durch eine eindeutige Position innerhalb genau eines Partitionselements repräsentiert.

Das Partitionsmuster wird beginnend an der Position des Versatzes wiederholt angewendet. Dadurch ist die Größe der Datei nicht auf die Länge des Partitionierungsmusters beschränkt, sondern kann beliebig wachsen.

Im Folgenden Beispiel werden anstatt einfacher nested FALLS die in der Darstellung kompakteren nested PITFALLS verwendet.

**Beispiel einer Dateipartitionierung** Abbildung 9 zeigt beispielhaft die Partitionierung einer Datei. Die Datei wird unter Verwendung des PITFALLS $(0, 1, -, 1, 2, 3, \emptyset)$ als Partitionierungsmuster und dem Versatz von 2 auf drei Partitionselemente aufgeteilt.

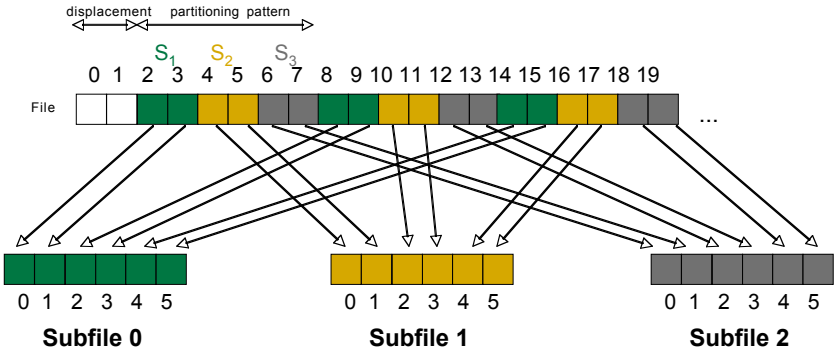


**Abbildung 9:** Beispiel der Partitionierung einer Datei mit dem Partitionierungsmuster des PITFALLS $(0, 1, -, 1, 2, 3, \emptyset)$ und dem Versatz 2.

Das PITFALLS Partitionierungsmuster $(0, 1, -, 1, 2, 3, \emptyset)$ lässt sich in die einzelnen nested FALLS zerlegen:

$$\begin{aligned} S_1 &= (0, 1, -, 1) \\ S_2 &= (2, 3, -, 1) \\ S_3 &= (4, 5, -, 1) \end{aligned}$$

Man beachte, dass die Positionen *relativ* zum Versatz (hier: 2) gesehen werden müssen. Die Datei wird in diesem Beispiel ringartig (*round robin*) auf die drei Subfiles aufgeteilt.

## 4.2 Physikalische Dateipartitionierung

Die auf die im vorangegangenen Abschnitt beschriebene Weise erstellten Subfiles können entweder auf einen einzelnen I/O-Knoten abgespeichert oder auf mehrere I/O-Knoten verteilt werden (*striping*).

Zwei Varianten müssen hier unterschieden werden: entweder es existieren mehr Subfiles als I/O-Knoten oder umgekehrt.

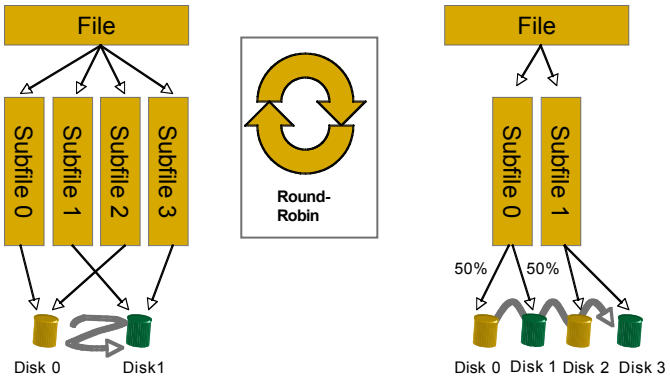


**Abbildung 10:** Aufteilung von Subfiles auf I/O-Knoten.

Falls mehr Subfiles als I/O-Knoten vorhanden sind, wird jedes Subfile auf aufeinander folgende I/O-Knoten geschrieben; falls man beim letzten I/O-Knoten angekommen ist, wird wieder mit dem ersten Knoten begonnen (round robin). Abbildung 10 zeigt auf der linken Seite die Verteilung von vier Subfiles auf zwei I/O-Knoten. Subfile 0 und 2 werden auf Disk 0, Subfile 1 und 3 werden auf Disk 1 abgelegt.

Im zweiten Fall existieren weniger Subfiles als I/O-Knoten (siehe rechte Seite der Abb. 10). Diese werden gleichmäßig auf die unterschiedlichen I/O-Knoten aufgeteilt. In dem Beispiel wird das Subfile 0 zu je 50 % auf Disk 0 und Disk 1, das Subfile 1 zu je 50 % auf Disk 2 und 3 gespeichert.

## 4.3 Logische Dateipartitionierung

Die logische Partitionierung von Dateien wird in Clusterfile durch die Definition von Views realisiert. Eine View ist ein Dateibereich, der scheinbar lineare Adressen besitzt und dessen Lade- bzw. Speichervorgang dementsprechend nur aus einem einzigen Aufruf besteht. Eine View ist mit einem Subfile vergleichbar, jedoch ist die Datei nicht derart gespeichert, wie sie für die Anwendung sichtbar ist. Wenn also die Datei – typischerweise – als Ganzes für die Anwendung sichtbar ist, kann sie dennoch auf mehreren I/O-Knoten verteilt gespeichert sein. Auch das Ein-/Ausgabekonzept von MPI (MPI-IO) verwendet Views. Bei Clusterfile wird für die Definition von Views wie bei den Subfiles auf die Repräsentation durch nested PITFALLS zurück gegriffen.

Der wichtigste Vorteil von Views ist die Entlastung des Programmierers von komplexen und fehleranfälligen Indexberechnungen. Einmal definiert, hat die Anwendung eine logisch nahtlose Sicht auf die benötigten Dateien und kann darauf in gewohnter Weise zugreifen.

Eine weitere positive Eigenschaft von Views ist die Tatsache, dass Mappings zwischen logischer und physischer Partitionierung im Voraus berechnet werden können und nicht erst im laufenden Betrieb. Die Zugriff-Indices werden einmalig bei der Definition einer neuen View berechnet und erlauben danach einen effizienten Zugriff auf die angeforderten Dateiabschnitte.

# 5 Mapping-Funktionen und Datenumverteilung

Das Dateimodell von Clusterfile erlaubt sowohl die logische als auch die physikalische Partitionierung von Dateien. Während eine *Datei* grundsätzlich eine zusammenhängende, linear adressierbare Folge von Bytes ist, lassen sich sowohl logische als auch physikalische *Partitionierungen* in Views bzw. Subfiles gleichermaßen als Tupel $(d, \mathcal{P})$ aus einem Versatz $d$ und einem Partitionierungsmuster $\mathcal{P}$ darstellen (s. Abschnitt 4.1).

Mapping-Funktionen dienen dazu, diese unterschiedlichen Partitionierungen möglichst effizient ineinander umzurechnen. Das ist immer dann notwendig, wenn Dateioperationen stattfinden, die logische Partitionierung jedoch nicht mit der physikalischen Partitionierung der betreffenden Datei übereinstimmt. Die Eigenschaft, dass das Darstellungsformat der Partitionierungen einheitlich ist, ermöglicht eine hohe Effizienz dieser Mapping-Funktionen. Der detaillierte algorithmische Aufbau der Mapping-Funktionen von Clusterfile wird in [IT2002] beschrieben.

Da die Performance paralleler Anwendungen in hohem Maße davon abhängt, ob die physikalische Verteilung der Daten auf die verschiedenen I/O-Knoten den Zugriffsmustern der Rechenknoten auf diese Daten entspricht, bietet Clusterfile die Möglichkeit einer Migration von einer physikalischen Verteilung in eine andere. Das kann *on-the-fly* geschehen, d.h. die Umverteilung kann nach einem Schema erfolgen, das sich erst zur Laufzeit als besser geeignet herausstellt. Angestoßen werden kann diese Umverteilung entweder von einem der Rechenknoten oder vom Betriebssystem der I/O-Knoten, das automatisch erkennt, ob eine Umverteilung der Daten angebracht ist, z.B. wenn es dauerhaft zu Serialisierungen paralleler Zugriffe auf den I/O-Knoten kommt. Die letztere Funktionalität von Clusterfile ist jedoch noch nicht implementiert (s. Abschnitt 7).

Durch die Gleichmäßigkeit der Verteilungsmuster, die sich aus der komprimierten Darstellungsweise als Nested PITFALLS (vgl. Abschnitt 3) ergibt, können auch physikalische Umverteilungen von multidimensionalen Feldern durch Clusterfile effizient durchgeführt werden. Darin geht Clusterfile über die Fähigkeiten anderer paralleler Dateisysteme hinaus, wie z.B. die des *Vesta Parallel File System*, das nur dann Partitionierungen von multidimensionalen Feldern unterstützt, wenn diese vorher in zweidimensionale Felder zerlegt werden ([IT2002]).

Der Umverteilungsalgorithmus von Clusterfile wird auch dazu verwendet, *Scatter-Gather-Operationen* auszuführen, die mit den Pack-And-Unpack-Methoden von *PVFS* und *MPI* vergleichbar sind.

## 5.1 Mapping-Funktionen

Die grundlegende Form einer Mapping-Funktion ist eine Abbildung eines Dateioffsets in eine Partition und einen Partitionsoffset. Die Funktion

$$(i, y) = MAP_{\mathcal{P},d}(x)$$

liefert für ein gegebenes Partitionierungsmuster $\mathcal{P}$, einen Versatz $d$ und einen Dateioffset $x$ die Nummer $i$ der Partition sowie den Offset $y$ innerhalb dieser Partition. Diese Funktionen sind injektiv und surjektiv, also insbesondere invertierbar und haben die in

Abschnitt 4.1 beschriebenen Eigenschaften. Eine Mapping-Funktion würde beispielsweise für den Dateioffset 7 aus Abbildung 11 das Tupel $(1, 3)$ zurückliefern. Das siebte Byte der Datei liegt also im Subfile 1, an Offset 3.

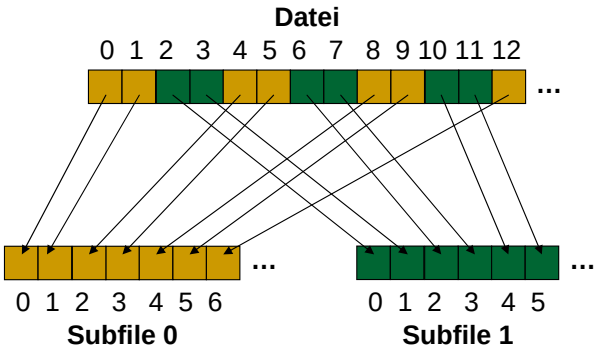


**Abbildung 11:** Mapping einer Datei auf ihre Subfiles durch eine Mapping-Funktion (Physikalische Partitionierung).

Die Umkehrung einer Mapping-Funktion hat die Form

$$x = MAP^{-1}_{\mathcal{P},d}(i, y)$$

und liefert für ein gegebenes Partitionierungsmuster $\mathcal{P}$, einen Versatz $d$, eine Partition $i$ und einen Partitionsoffset $y$ die entsprechende Position $x$ innerhalb der Datei zurück.

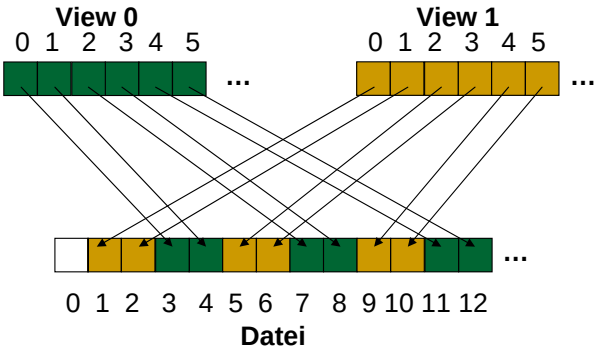


**Abbildung 12:** Mapping von Views auf eine Datei durch eine inverse Mapping-Funktion (Logische Partitionierung).

Verkettungen von Mapping-Funktionen können dazu dienen, um Elemente zweier verschiedener Partitionierungen direkt ineinander umzurechnen (*direktes Mapping*).

Da die logische Partitionierung durch eine Anwendung nicht unbedingt mit der physikalischen Partitionierung auf den I/O-Knoten übereinstimmt, wird jedes Mal, sobald eine Anwendung einen View auf die Daten erzeugt, ein Mapping zwischen dem View auf die Datei einerseits und der physikalischen Verteilung der Datei andererseits berechnet. Seien $\mathcal{P}_1$ das Partitionierungsmuster und $d_1$ der Versatz eines Views. Für einen Offset

$x$ eines Abschnitts $i$ dieses Views berechnet sich die physikalische Adresse nach dem Schema

$$(j, x) = MAP_{\mathcal{P}_2,d_2}(MAP^{-1}_{\mathcal{P}_1,d_1}(i, y))$$

$j$ ist dann die Nummer des entsprechenden Subfiles und $x$ der Offset in diesem Subfile.

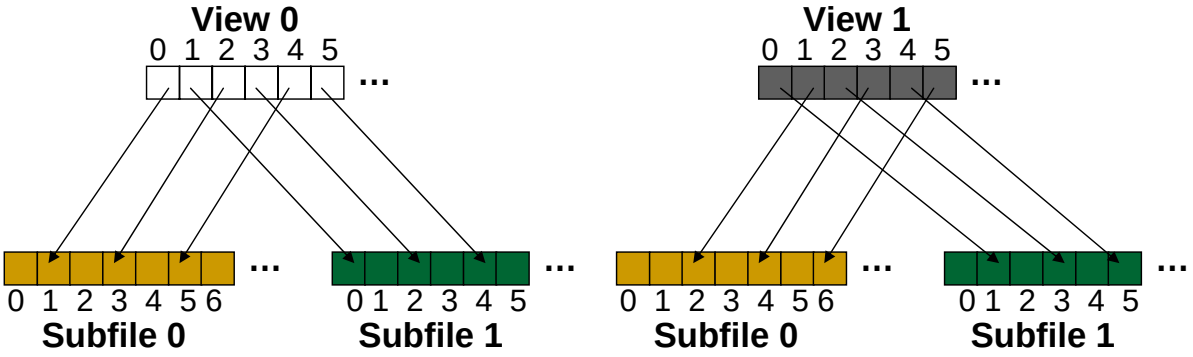


**Abbildung 13:** Direktes Mapping von Views auf Subfiles durch eine Verkettung von Mapping-Funktionen.

Der Fall, in dem die logische und die physikalische Partitionierung einer Datei übereinstimmen, stellt in zweierlei Hinsicht ein Optimum dar. Zum einen muss keine Mapping-Funktion berechnet werden, weil diese die Identität ist. Andererseits führt ein Zugriff auf zusammenhängende Daten auf Applikationsebene automatisch zu einem Zugriff auf zusammenhängende Daten auf den Datenträgern. Es müssen nur wenige Nachrichten auf dem Netzwerk erzeugt werden, und die Dateioperationen können durchgeführt werden ohne Verzögerungen durch Neupositionierungen der Schreib-/Leseköpfe in den Festplatten.

Wenn die logische und die physikalische Partitionierung einer Datei unterschiedlich ist, kann es bei einem direkten Mapping zwischen View und Subfiles passieren, dass zahlreiche kleine Netzwerkanfragen verschickt werden müssen. Aus diesem Grund wird wird von Clusterfile auch ein *getrenntes Mapping* (*Split Mapping*) unterstützt, das immer dann sinnvoll ist, wenn mehrere nicht zusammenhängende Abschnitte eines Views oder eines Subfiles zwischen einem Rechenknoten und einem I/O-Knoten ausgetauscht werden müssen.

Wie in den Abbildungen 14 und 15 dargestellt, werden für jeden neu erzeugten View auf dem entsprechenden Rechenknoten zunächst die Schnittmengen dieses Views mit den einzelnen Subfiles berechnet. Diese Schnittmengen enthalten diejenigen Bereiche der Datei, die sowohl vom neu erzeugten View als auch vom entsprechenden Subfile abgedeckt werden. Auf Basis dieser Schnittmenge werden anschließend für jedes View-Subfile-Paar zwei Projektionsfunktionen berechnet: Eine *View-Projektion* (*View-Mapping*), die den linearen Bereich eines Views auf einen linearen Puffer im Rechenknoten abbildet, sowie eine *Subfile-Projektion* (*Subfile-Mapping*), die den Puffer im I/O-Knoten abbildet auf ein Subfile. Diese Funktionen lassen sich in kompakter Form als Nested PITFALLS darstellen. Die Subfile-Projektionen werden nach ihrer Erstellung von den Rechenknoten an

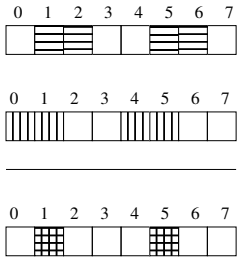


**Abbildung 14:** Schnittmengenbestimmung eines Views und eines Subfiles.

die betroffenen I/O-Knoten kommuniziert. Bei einem Schreibzugriff werden die einzelnen zu schreibenden Fragmente eines Subfiles vom Rechenknoten nicht separat an den I/O-Knoten geschickt. Stattdessen werden diese mit Hilfe der View-Projektion in einem auf dem Rechenknoten vorhandenen Puffer gesammelt (*Gather*) und dann zusammen verschickt. Der entsprechende I/O-Knoten empfängt das Datenpaket in einem eigenen Puffer und bedient sich der Subfile-Projektion, um die Daten im Puffer auf das Subfile zu verteilen (*Scatter*).

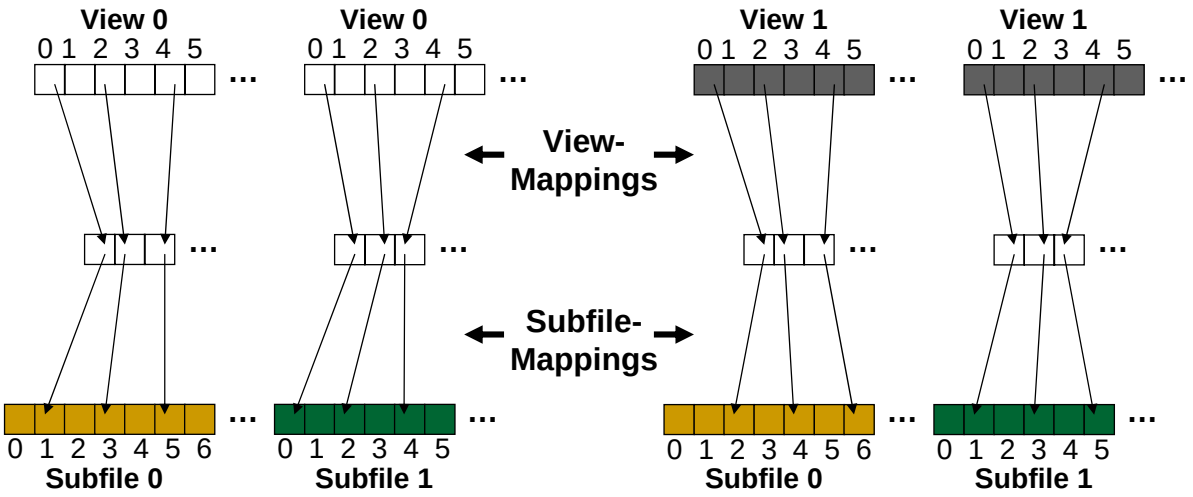


**Abbildung 15:** Split Mapping von Views auf den linearen Dateibereich und vom linearen Dateibereich auf die Subfiles.

## 5.2 Datenumverteilung

Um eine physikalische Datenumverteilung vorzunehmen, muss zunächst ermittelt werden, wie die Dateidaten in der Anfangsverteilung auf die Subfiles der Zielverteilung verschoben werden müssen. Um diese Abbildung zu erstellen, verfährt Clusterfile wie beim Mapping zwischen einem View und einem Subfile. Für jedes Paar $(p_1, p_2)$ eines Subfiles $p_1$ aus der Anfangsverteilung und eines Subfiles $p_2$ aus der Zielverteilung wird zunächst die Schnittmenge dieser beiden Subfiles berechnet, d. h. die Menge von Daten

der eigentlichen Datei, die in beiden Subfiles vorkommen. Das Ergebnis dieses Schnittoperation ist wiederum ein Nested PITFALLS. Aus diesem werden anschließend zwei bijektive Projektionen zwischen dem linearen Bereich der Datei und der Anfangsverteilung bzw. der Zielverteilung erstellt, und die Daten werden entsprechend auf den I/O-Knoten verschoben.

# 6 Komponenten und Dateioperationen

Clusterfile besteht aus den drei Komponenten Metadaten-Manager, I/O-Server und I/O-Clients, die im folgenden vorgestellt werden.

**Metadaten-Manager** Der Metadaten-Manger verwaltet die Metadaten des Dateisystems. Dazu gehören Dateiattribute wie Dateityp, Dateigröße, Zeitpunkt der Erstellung, Berechtigungen usw., aber auch Informationen über die physikalische Verteilung der Dateien, d. h. Informationen darüber, wie die Dateien partitioniert sind, und auf welchen I/O-Knoten die einzelnen Abschnitte gespeichert sind. Die Informationen sammelt der Metadaten-Manager selbst in regelmäßigen Abständen oder auf Anfrage von den I/O-Knoten und sorgt für die Konsistenz dieser Informationen. Er selbst speichert keine Dateidaten und unterstützt die Rechenknoten bei Operationen, die auf Metadaten durchgeführt werden, wie z. B. das Generieren von Mapping-Funktionen. Bei den eigentlichen Dateioperationen (Lesen, Schreiben) spielt der Metadaten-Manager keine Rolle. Im derzeitigen Entwicklungsstand von Clusterfile ist nur ein einziger Metadaten-Manager vorgesehen, weshalb dieser bei einer Häufung von Zugriffen auf kleine Dateien zu einem limitierenden Faktor werden kann. In [I+2004] werden Erweiterungen von Clusterfile vorgestellt, die dieses Problem durch Caching von Dateidaten auf den Rechenknoten reduzieren sollen.

**I/O-Server** Auf jedem I/O-Knoten des Clusterfile-Dateisystems läuft ein I/O-Server, der sich hauptsächlich um das Schreiben in die Subfiles und das Lesen aus den Subfiles kümmert. Um die von den Clients empfangenen Datenpakete den lokalen Subfiles bzw. Subfile-Fragmenten zuordnen zu können, werden bei jeder Erstellung eines Views auf einem Rechenknoten Subfile-Projektionen an alle I/O-Server übertragen, die Daten dieses Views in Subfiles speichern (vgl. Abschnitt 5.1). Die I/O-Server nutzen die Subfile-Projektionen, um beim Eingang einer Schreib- oder Leseaufforderung die Daten im Empfangspuffer auf die Subfiles zu verteilen, bzw. beim Lesen die Daten aus den Subfiles im linearen Sendepuffer zu sammeln. Die durch die Dateioperationen anfallenden Metadaten, wie Dateigröße, Zeitpunkt der Erstellung usw., werden von den I/O-Servern zwischengespeichert und auf Anfrage an den Metadaten-Manager übertragen.

**I/O-Clients** Die I/O-Clients bedienen sich der Funktionen einer *Clusterfile-I/O-Bibliothek*, um Dateioperationen durchzuführen. Dabei ist die Architektur von Clusterfile

**Listing 1:** Beispiel eines Clusterfile-Dateizugriffs.

```
1  main() {
2  struct pitfalls *phys_dist=create_pitfalls(0,0,2,8,1,2,NULL);
3  struct pitfalls *view=
4      create_pitfalls(4*proc_id,4*(proc_id+1)-1,-1,1,-1,1,NULL);
5  int fd = open("filename",O_CREAT|O_RDWR,0x666);
6  ioctl(fd,SET_PHYSICAL_DISTRIBUTION,phys_dist);
7  ioctl(fd,SET_VIEW,view);
8  write(fd,buf,4);
9  close(fd);
10 }
```

für die I/O-Clients transparent; diese müssen nicht explizit mit dem Metadaten-Manager oder den I/O-Servern kommunizieren. Listing 1 zeigt beispielhaft die Programmierung eines Schreibzugriffs auf eine Clusterfile-Datei. Da Clusterfile unter UNIX als Standard-Dateisystemschnittstelle implementiert ist, können nach dem Öffnen der Datei durch einfache `ioctl`-Kommandos physikalische Partitionierungen erzeugt und Views erzeugt werden. Durch das `SET_PHYSICAL_DISTRIBUTION`-Kommando werden die Dateidaten gemäß dem neuen Layout neu partitioniert. Die Informationen über das neue Layout werden dann von den I/O-Knoten an den Metadaten-Manager gesendet.

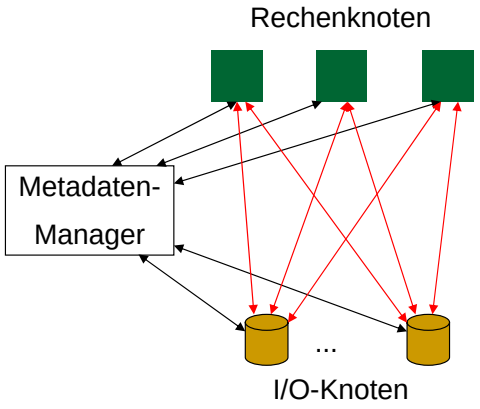


**Abbildung 16:** Architektur von Clusterfile. Datenströme sind rot, Metadatenströme schwarz eingezeichnet.

Insgesamt stellt sich die Architektur von Clusterfile wie in Abb. 16 dar. Metadaten werden zwischen I/O-Knoten, Rechenknoten und Metadaten-Manager ausgetauscht, während die Nutzdaten nur zwischen den Rechenknoten und den I/O-Knoten kommuniziert werden.

Sowohl Lese- als auch Schreiboperationen finden in zwei Phasen statt. Die erste Phase findet nach jedem Setzen eines neuen Views auf den Rechenknoten statt. In ihr wer-

den View- und Subfile-Projektionen vom Clusterfile-Subsystem auf den Rechenknoten erzeugt. Die zweite Phase umfasst folgende Schritte. Das Lesen erfolgt in umgekehrter Reihenfolge.

1. Die in der ersten Phase berechneten Projektionen werden verwendet, um für jedes betroffene Subfile die in dieses Subfile zu schreibenden Daten in einem Schreibpuffer zu sammeln.

2. Der Inhalt des Puffers wird an den entsprechenden I/O-Knoten gesendet.

3. Der I/O-Knoten empfängt die Daten zunächst in einem Puffer und verteilt sie mit Hilfe der Subfile-Projektion auf die Subfiles.

4. Die neuen Metadaten werden vom I/O-Knoten an den Metadaten-Manager gesendet.

# 7 Zusammenfassung und Ausblick

Clusterfile ist ein paralleles Dateisystem, das den Anwendungen ein hohes Maß an Flexibilität bei der Verteilung von Daten auf Rechencluster bietet. Dabei muss nicht unbedingt im Voraus eine Festlegung über die Verteilung getroffen werden, sondern diese kann bei Bedarf auch später ohne Datenverlust geändert werden. Auch die logische Partitionierung von Daten ist flexibel handhabbar. Der Programmierer wird von fehleranfälligen Indexberechnungen entlastet, und der Sourcecode bleibt besser verständlich. Die Algorithmen, die Clusterfile verwendet, profitieren von der kompakten und einheitlichen Darstellung von Views, Subfiles, Schnittmengen und Projektionen. Auch multidimensionale Felder können dank der beliebig tief verschachtelbaren PITFALLS dargestellt und verarbeitet werden. Durch die Vorberechnungen von Projektionen bei der Erstellung eines Views wird die Umrechnung einer logischen auf eine physikalische Partitionierung, die bei den Dateioperationen ggf. notwendig ist, stark beschleunigt. Sowohl die Datenumverteilung on-the-fly als auch das Verändern eines Views auf bereits gespeicherte Daten ist zum Zeitpunkt der Ausführung teuer. Dieser Aufwand kann sich jedoch bei fortschreitenden Dateizugriffen amortisieren.

Als Ausblick nennen die Entwickler in [IT2003] die folgenden Punkte:

**Kernel-Implementierung** Im Moment ist die Implementierung von Clusterfile noch auf den Userspace beschränkt. Eine Implementierung im Kernel ist fest vorgesehen und soll unnötigen Overhead vermeiden, der durch die Kommunikation zwischen Userspace und Kernelspace entsteht.

**Kollektive I/O-Operationen** Um den gleichzeitigen Zugriff mehrerer Rechenknoten auf die selbe Datei zu beschleunigen, sollen mehrere Dateizugriffe in kollektiven Zugriffen zusammengefasst werden können.

**Abstraktes Kommunikationsinterface** Statt wie die derzeitige Implementierung auf TCP/IP-Sockets zurückzugreifen, könnte eine Low-Level-Schnittstelle wie *GM* (*Glenn's*

*Messages* - Ein Nachrichten-basiertes Kommunikationssystem für Myrinet) die Performance von Clusterfile weiter erhöhen.

**Kooperative Caches** Die I/O-Knoten könnten entlastet werden, indem Dateianfragen durch Caches auf den Rechenknoten bedient werden.

**Skalierbarkeit des Metadaten-Managers** Die Gefahr einer Überlastung des Metadaten-Managers könnte durch die Möglichkeit, weitere Metadaten-Manager zu integrieren, minimiert werden. Dann müssten Kohärenzprotokolle die Integrität der Metadaten sichern.

Des weiteren wäre das automatische Anstoßen einer physikalischen Umverteilung der Daten auf den I/O-Nodes sinnvoll. Dann könnte eine optimale physikalische Verteilung auch für solche Anwendungen erreicht werden, die nicht speziell auf das Clusterfile-Dateisystem ausgelegt sind.

## Literatur


[IT2002] Isaila, Florin und Walter F. Tichy: Mapping Functions and Data Redistribution for Parallel Files, Proceedings of IPDPS 2002 Workshop on Parallel and Distributed Scientific and Engineering Computing with Applications, Fort Lauderdale, April 2002. PDF: `http://www.ipd.uka.de/~florin/Publications/final.pdf`.

[IT2003] Isaila, Florin und Walter F. Tichy: Clusterfile: A Flexible Physical Layout Parallel File System, Concurrency & Computation: Practice & Experience, John Wiley & Sons Ltd, Volume 15, Issue 7-8 (June - July 2003), pages 653-679. PDF: `http://www.ipd.uka.de/~florin/Publications/PAPER.pdf`.

[I+2004] Isaila, Florin et al.: Integrating Collective I/O and Cooperative Caching into the Clusterfile Parallel File System, ICS '04 (June - July 2004).