

Einführung in die Betriebssysteme

Ein- und Ausgabe von Dateien Teil I

Michael Rennecke

Dozenten:

Prof. Dr. Paul Molitor

Dipl.-Inform. Annett Thüring

Halle, 2006

Inhaltsverzeichnis

Inhaltsverzeichnis	i
1 RAID-Systeme	1
1.1 Geschichte	1
1.2 Einstieg	1
1.3 Voraussetzungen und Ziele	2
1.4 RAID-Level	3
1.4.1 RAID 0 oder Data Striping	3
1.4.2 RAID 1 oder Drive Mirroring / Drive Duplexing	3
1.4.3 RAID 2 oder Hamming-System	4
1.4.4 RAID 3 oder Byte Striping mit Parity Laufwerk	4
1.4.5 RAID 4 oder Block Striping mit Parity Laufwerk	5
1.4.6 RAID 5 oder Block Striping mit verteilter Parity	5
1.4.7 RAID 6 Block oder Striping mit verteilter Parity auf 2 Platten	6
1.4.8 NRAID oder Festplattenverbund	7
1.4.9 Kombinierte RAID-Level	7
1.5 Datenrettung	8
1.5.1 Hot-Plugging ohne Spare-Disk	8
1.5.2 Hot-Plugging mit Spare-Disk	9
1.6 Hardware-RAID	9
1.7 Software-RAID	9
1.8 Nutzen von RAID	10
2 Dateiverwaltung in Multiuser-Systemen	11
2.1 Einstieg	11
2.2 Arten von Dateisystemen	12
2.2.1 Lineare Dateisysteme	12
2.2.2 Hierarchische Dateisysteme	12
2.2.3 Netzwerkdateisysteme	12
2.2.4 Datenbank-Dateisysteme	13

2.2.5	Spezielle virtuelle Dateisysteme	13
2.3	Dateiorganisation	14
2.3.1	Dateiallokation	14
2.3.2	Cluster	16
2.4	Dateiverzeichnisse	17
2.5	Zugriffsrechte	18
2.6	Das Dateisystem ext3	19
2.6.1	Reservierter Speicherplatz	20
2.6.2	Datensicherheit	20
	Liste der Abkürzungen	21

Kapitel 1

RAID-Systeme

1.1 Geschichte

Festplatten waren in den Anfangszeiten der Datenverarbeitung sehr teuer. Es gab eine Zweiklassengesellschaft für Festplatten. Die kostengünstigen waren für PCs und einfache Server vorgesehen, sie hatten jedoch einen großen Nachteil: sie fielen häufiger aus. Dies war gerade für den Großrechnerbereich und wichtige Server nicht tolerierbar, deshalb wurden für diesen Einsatz speziell gefertigte und kontrollierte Festplatten verwendet. Diese waren sehr teuer und wurden nur von einzelnen Herstellern verwendet.

Als Lösung veröffentlichten 1987 D. A. Patterson, G. Gibson und R. H. Katz von der University of California, Berkeley, USA einen Vorschlag (RAID), um die langsamen Plattenzugriffe zu beschleunigen und die MTBF (*Mean time between failures*) zu erhöhen. Dazu sollten die Daten auf vielen kleineren (günstigeren) Platten anstatt auf wenigen großen (teuren) abgelegt werden. Deshalb hieß die frühere Schreibweise auch Arrays of Inexpensive Disks (heute Independent), im Gegensatz zu den damaligen SLEDs (*Single Large Expensive Disk*). Außerdem war die Wahrscheinlichkeit, dass mehrere günstige Festplatten ausfallen geringer, wie der Ausfall einer SLED.

Dieses Verfahren löste die beiden wichtigsten Nachteile der Festplatten: den Preis und die Ausfallwahrscheinlichkeit.

1.2 Einstieg

Früher war RAID die Abkürzung von **R**edundant **A**rray of **I**nexpensive **D**isks (*Redundante Anordnung preiswerter Festplatten*) übersetzt, heute sagt man eigentlich, dass RAID für **R**edundant **A**rray of **I**ndependent **D**isks (*Redundante*

Anordnung unabhängiger Festplatten) steht. Die Gründe für die zwei Abkürzungen sind von geschichtlicher Natur. Heute werden bei RAID mehrere physische Festplatten zu einem großen logischen Laufwerk vereint. Den Zusammenschluss von mehreren Festplatten nennt man hier Array. In diesem Array werden nicht nur Daten, sondern auch redundante Informationen gespeichert. Mit Hilfe dieser Redundanz kann man bei dem Ausfall einer HD die Daten rekonstruieren. Aus der Sicht des Benutzers oder eines Programmes unterscheidet sich ein logisches RAID-Laufwerk nicht von einer Festplatte bzw. eine Partition.

Diese Redundanten Informationen können wie folgt aussehen: Wenn es die Daten selbst sind spricht man von spiegeln, andernfalls werden Parity Daten aus mehreren Datenblöcken berechnet.

1.3 Voraussetzungen und Ziele

Für den Betrieb eines RAID-Systemes werden im Normalfall mindestens 2 Festplatten* benötigt. In modernen Betriebssystemen kann man ein RAID-System auch mit einer Festplatte, die entsprechend partitioniert ist, simulieren. Dadurch gehen natürlich die Vorteile eines RAID-Systemes verloren. Die Festplatten oder die Partitionen müssen bei RAID immer die gleiche Größe haben, außer bei NRAID.

Wenn man einen RAID-Verbund von mehreren Festplatten hat erhält man folgende Vorteile:

- Erhöhung der Datensicherheit (Redundanz)
- Steigerung der Transferraten (Performance)
- Aufbau großer logischer Laufwerke
- Austausch von Festplatten und Erhöhung der Speicherkapazität während des Systembetriebes
- Kostenreduktion durch Einsatz mehrerer preiswerter Festplatten
- schnelle Steigerung der Systemleistungsfähigkeit

Man muss aber bedenken, dass RAID nur vor einen Festplattenausfall schützt und nicht vor dem Löschen von Daten oder den Diebstahl/Zerstörung von Festplatten. Deshalb ist immer noch ein Backup der Daten notwendig, wenn man sich vor diesen Eventualitäten schützen möchte.

*Die Mindestanzahl von HD's hängt auch vom RAID-Level ab

1.4 RAID-Level

1.4.1 RAID 0 oder Data Striping

Bei RAID 0 handelt es sich nach der Definition von RAID, nicht um ein RAID-System, da hier die Redundanz fehlt. Für diesen RAID-Level werden die verschiedenen Festplatten in Arrays zusammen geschaltet. Dadurch kann man die Schreib- bzw. Leseoperationen parallel ausführen. Das wird dadurch erreicht, dass die Daten in Stücke geteilt werden (*vgl. stripping engl. in Streifen zerlegen*). Diese Performance-Steigerung macht sich vor allem bei sequentiellen Zugriffen bemerkbar. Beim schreiben werden die Daten erst auf die Caches der verschiedenen Festplatten verteilt, danach werden diese auf die Festplatten geschrieben. Wenn man liest wird der Schreibvorgang umgekehrt ausgeführt. Durch das Verteilen eines Datenstücks auf mehrere Festplatten, erreicht man, dass die einzelne Festplatte weniger zu arbeiten hat.

Wenn bei RAID 0 eine Festplatte ausfällt, sind alle Daten zerstört. Die Ausfallwahrscheinlichkeit steigt linear mit der Anzahl der verwendeten Festplatten. Dadurch ist RAID 0 für den Servereinsatz völlig unbrauchbar.

Dieser Level kam erst später dazu, d.h. man hat ihn nicht benutzt um viele kleine Festplatten zu einen großen logischen Laufwerk zu verschalten. Der Grund ist fehlende Redundanz.

Kapazität	‡ Festplatten * Kapazität (der kleinsten Festplatte)
Geschwindigkeit	sehr hoch
Ausfallwahrscheinlichkeit	sehr hoch
Kosten	sehr gering, da volle Auslastung der Festplatten
Anwendungen	Video-Schnitt, Temporärspeicher

1.4.2 RAID 1 oder Drive Mirroring / Drive Duplexing

Ein RAID-1-Array besteht aus zwei oder mehr Festplatten, die dieselben Daten enthalten. In der Regel sind das zwei Festplatten, es ist aber auch möglich, mehr im Array zu haben. RAID 1 bietet die volle Redundanz der gespeicherten Daten.

Fällt eine Festplatte aus werden die Daten von der anderen Festplatte gelesen und geschrieben. Das ist bei Echtzeit-Anwendungen sehr wichtig. Den Totalverlust der Daten erleidet man erst, wenn beide Festplatten[†] ausfallen.

Wenn alle Festplatten am selben Controller angeschlossen sind spricht man von Mirroring (*Spiegelung der Daten*). Da immer nur einen Zugriff auf den Controller geben kann, ist die Redundanz eines Sektors nicht sofort gegeben. Sie ist erst gegeben, wenn der Sektor auf einer zweiten Festplatte geschrieben wurde,

[†]Die Festplatte mit Daten und die mit den entsprechenden Redundanzdaten

d.h. man kann max. einen Sektor verlieren (bei Ausfall einer Platte). Wenn man die Festplatten an verschiedene Controller anschließt ist die Redundanz sofort gegeben und man spricht von Duplexing.

RAID 1 kann eine erhöhte Performance beim Lesen bewirken, weil Daten von einer Festplatte angefordert werden können, während die andere noch beschäftigt ist.

Kapazität	$\frac{1}{2} * \# \text{ Festplatten} * \text{Kapazität}$
Geschwindigkeit	hoch
Ausfallwahrscheinlichkeit	gering, eine HD darf ausfallen ohne Datenverlust
Kosten	sehr hoch, da halbe Auslastung der Festplatten
Anwendungen	Datenbanken, Betriebssysteme, hohe I/O-Last

1.4.3 RAID 2 oder Hamming-System

Zu diesen RAID-Level gibt es eigentlich sehr viel zu sagen, da er in der Praxis keine Rolle mehr spielt, möchte ich diesen Abschnitt nicht übermäßig viel Gewicht geben.

Die Daten werden hierbei in Bitfolgen fester Größe zerlegt und mittels des Hamming-Codes auf größere Bitfolgen abgebildet (8 Bit für Daten[‡] noch 3 Bit für den ECC-Code (*Error Correction Code*)). Der ECC wird nach dem Hamming-Algorithmus berechnet und auf zusätzlichen Festplatten gespeichert und die Datenbits werden auch auf verschiedene Festplatten verteilt. Dadurch erreicht man prinzipiell einen hohen Datendurchsatz, aber die Anzahl der Festplatten muss ein Vielfaches der Hamming-Codewortlänge sein.

Dieses Verfahren wurde während der Anfänge von RAID in Großrechnersystemen verwendet, als die Festplatten noch keine integrierten ECC Mechanismen hatten. Das heutzutage in (allen)[§] Massenspeichern integrierte ECC Verfahren macht die Eigenschaft von RAID 2 Ein-Bit-Fehler korrigieren zu können überflüssig. Deshalb wird dieser RAID Level nicht mehr eingesetzt.

1.4.4 RAID 3 oder Byte Striping mit Parity Laufwerk

RAID 3 ist der Vorläufer des RAID 5. In einer RAID 3 Konfiguration werden die Daten in einzelne Bytes aufgeteilt, dann abwechselnd auf die Datenlaufwerke des Arrays geschrieben (üblicherweise 2-4 Festplatten) und die Redundanz wird auf einer zusätzlichen Festplatte gespeichert. Als Redundanz bezeichnet man hier das bitweise XOR (*Antivalenz*) der einzelnen Bits der anderen Festplatten.

[‡]Bei 8 Datenbits ist auch das Paritätsbit mitgezählt

[§]Mir sind keine HD's bekannt, die kein ECC haben, möchte aber meine Aussage nicht pauschalisieren

Leistungsverhalten

Angenommen das RAID 3-Array besteht aus n datentragenden Festplatten und einer dedizierten Parity-HD mit der Nummer $n + 1$. Bei einem Ausfall einer der ersten n Festplatten werden, nach Austausch dieser Festplatte, alle Daten auf die neue n -te Festplatte zurück synchronisiert, ausgehend von den Parity-Informationen der $n + 1$ -ten Festplatte und den übrigen, unversehrten $n - 1$ Daten-Festplatten. Der Ausfall der Parity-Disk selbst wird, nach Ersetzen der Festplatte, repariert durch einfache Neuberechnung der Parity-Bits.

RAID 3 ist fast vollständig vom Markt verschwunden und wurde durch RAID 5 ersetzt. Außerdem sei angemerkt, dass RAID 3 mit nur 2 Festplatten nach Definition identisch mit RAID 1 ist.

1.4.5 RAID 4 oder Block Striping mit Parity Laufwerk

RAID 4 ist mit RAID 3 vergleichbar, es werden Blöcke statt Bytes geschrieben/gelesen. Die Blöcke (*Chunks*) können 8, 16, 32, 64 oder 128 kByte groß sein. Der Parity-Block wird durch blockweise XOR der Datenblöcke berechnet. Die Rekonstruktion der Daten ist wie bei RAID 3. Der größte Vorteil von RAID 4 ist die Erweiterbarkeit um zusätzliche Festplatten (*HD muss beim Einbau nur Nullen enthalten*) ohne Umorganisation der Festplatteninhalte. Die Performance eines RAID 4 Systems ist sehr gut, wenn man große sequentielle Lese- und Schreibzugriffe hat (Schreiben großer zusammenhängender Dateien). Bei verteilten Schreibzugriffen wird durch jeden Benutzerzugriff ein Zugriff auf den zugehörigen Parity-Block notwendig. Das bedeutet, dass bei verteilten Zugriffen durch den Benutzer jedes Mal gewartet werden muss, bis die Parity Daten auf die Parity-HD geschrieben worden sind. Daraus resultiert, dass RAID 4 bei solchen verteilten Schreibzugriffen relativ langsam ist. Ein großer Nachteil sind Schreiboperationen, da die Parity-Informationen berechnet und auf die eine Parity-HD geschrieben werden müssen. Diese Platte wird schnell zum Flaschenhals im Gesamtsystem und fällt evtl. früher aus. Wegen der fest definierten Paritätsplatte wird statt RAID 4 fast immer RAID 5 bevorzugt.

1.4.6 RAID 5 oder Block Striping mit verteilter Parity

RAID 5 bietet gesteigerte Performance beim Lesen von Daten als auch Redundanz bei relativ geringen Kosten und ist dadurch die beliebteste RAID-Variante. In schreibintensiven Umgebungen ist RAID 5 nicht zu empfehlen, da sowohl bei sequenziellen als auch bei zufälligen Schreibzugriffen die Performance deutlich abnimmt. RAID 5 ist die kostengünstigste Möglichkeit Daten redundant auf 3 Festplatten zu speichern.

Bei RAID 5 ist die Datensicherheit des Arrays beim Ausfall von maximal einer Platte gewährleistet. Allerdings lässt nach Ausfall einer Festplatte oder während

der Rekonstruktion auf die Spare-Disk[¶] (bzw. nach Austausch der defekten Festplatte) die Performance deutlich nach. Die Rekonstruktion dauert länger als bei RAID 1, da bei RAID 5 zusätzlich Parity Informationen rekonstruiert werden müssen. Je mehr Festplatten in einem RAID 5 Verbund sind, desto länger dauert die Rekonstruktion bzw. desto schlechter ist die Performance während eines Defekts einer Festplatte.

Kapazität	(# Festplatten - 1) * Kapazität
Geschwindigkeit	normal
Ausfallwahrscheinlichkeit	gering, eine HD darf ausfallen ohne Datenverlust
Kosten	gering, nur eine HD Verlust
Anwendungen	Fileserver, Testsystem, Archivierung, Sever

Rechenbeispiele, nutzbare Gesamtkapazität in Prozent

Alle Festplatte haben 100 GByte Kapazität. Die nutzbare Gesamtkapazität in Prozent berechnet sich wie folgt: $100 - \frac{100}{n}$ Dabei ist n die Anzahl der HD's ohne Spare-Disks.

# HD's	Gesamtkap.	nutzbare Gesamtkap. in Prozent	nutzbare Gesamtkap.
3	300 GByte	$66\frac{2}{3}$	200 GByte
4	400 GByte	75	300 GByte
5	500 GByte	80	400 GByte
6	600 GByte	$80\frac{1}{3}$	500 GByte

1.4.7 RAID 6 Block oder Striping mit verteilter Parity auf 2 Platten

Dies ist eine Weiterentwicklung von RAID 5. Es können 2 Festplatten aus dem Verband ausfallen und die Daten sind immer noch lesbar. Je nach Implementation wird die Parity einfach wie bei RAID 5 berechnet und dann auf 2 Platten geschrieben, oder es wird eine neue Parity berechnet, inklusive der ersten Parity-Informationen. Bei den Lesezugriffen wird eine ähnliche Geschwindigkeit wie bei RAID 5 erreicht. Wenn eine spezielle Hardware zur Berechnung der Parity-Informationen verwendet wird, ist RAID 6 im schreibenden Zugriff ähnlich schnell wie RAID 5, sonst ist der schreibende Zugriff deutlich langsamer.

[¶]Die (Hot)spare-Disk ist eine unbenutzte Reservefestplatte. Fällt eine HD innerhalb des RAID-Verbundes aus, wird sie durch das Reservelaufwerk ersetzt. Dadurch bleibt die Redundanz erhalten.

Kapazität	$(\# \text{ Festplatten} - 2) * \text{Kapazität}$
Geschwindigkeit	normal
Ausfallwahrscheinlichkeit	sehr gering, zwei HD's dürfen ausfallen ohne Datenverlust
Kosten	normal, zwei HD's Verlust
Anwendungen	Fileserver, Testsystem, Archivierung, Sever

1.4.8 NRAID oder Festplattenverbund

Bei NRAID werden, wie bei RAID 0, mehrere Festplatten zusammengeschlossen. Im Gegensatz zu RAID 0 bietet NRAID aber keinen Performance-Gewinn. Dafür kann man Festplatten unterschiedlicher Größe ohne Speicherverlust miteinander kombinieren (Beispiel: eine 10 GByte-Festplatte und eine 30 GByte-Festplatte ergeben in einem NRAID eine virtuelle 40 GB Festplatte, während in einem RAID 0 nur 20 GByte (2 x 10 GByte) angesprochen werden könnten). Der Ausfall einer Platte führt zu Datenverlust, jedoch wäre es möglich einen Teil der Daten wieder zu restaurieren, solange sie komplett auf der funktionierenden Platte liegen. NRAID ist weder einer der nummerierten RAID-Levels, noch bietet es Redundanz. Man kann es aber durchaus als entfernten Verwandten von RAID 0 betrachten. NRAID macht aus mehreren Festplatten eine einzige Partition; einer, die der Summe der Kapazitäten aller verwendeten Platten entspricht. Dies ist auch als *linear mode* bekannt.

1.4.9 Kombinierte RAID-Level

Hier versucht man verschiedene RAID-Level in einer Baumstruktur anzuordnen um Vorteile zu erreichen. Diese Kombinations-RAID-Level haben in der Regel keine herausragend gute Eigenschaften. In der Praxis trifft man die RAID-Level 0, 1 und 5 man meisten an.

Von den Kombinations-RAID werden RAID 01 und RAID 10 am meisten verwendet.

Schreibweise

Wenn zwei RAID 5-Arrays zu einen RAID 1-Array kombiniert werden: RAID 5+1 oder RAID 51. Wenn umgekehrt zwei RAID 1-Arrays zu einen RAID 5-Array verschaltet werden: RAID 1+5 bzw. RAID 15.

RAID 01 und RAID 10 oder Mirrored Striped RAID

Dieser Raidlevel wurde erst später geschaffen. Er ist eine Kombination aus Raid-level 1 und 0, also dem Spiegeln der Daten und dem Aneinanderreihen der Festplatten. Sie verbinden den Geschwindigkeitsvorteil von Raid 0 und die Sicherheit von Raid 1. Der Einsatz bietet sich speziell bei Datenbanken an.

Kapazität	$\frac{1}{2} * \# \text{ Festplatten} * \text{Kapazität}$
Geschwindigkeit	sehr hoch
Ausfallwahrscheinlichkeit	gering, eine HD darf ausfallen ohne Datenverlust
Kosten	hoch nur halbe Auslastung
Anwendungen	Datenbanken, hohe I/O-Last

1.5 Datenrettung

Dieser Teil gilt nur für RAID-Level, die Redundanzen haben. Wenn man keine Redundanzen hat, dann kann man auch keine Daten rekonstruieren oder man schlägt Wege ein, welche nicht RAID-spezifisch sind.

Generell ist zu sagen, dass man eine defekte HD immer gegen eine intakte ersetzen muss um die korrekte Arbeitsweise des RAID-Systems zu gewährleisten. Es besteht z.B. bei RAID 1 die Möglichkeit, wenn eine HD ausfällt, nur noch die Intakte zu benutzen.

Die nachfolgenden Vorgehensweisen beschreiben Methoden, wenn man den Computer nicht herunterfahren kann, um die defekte HD zu ersetzen. Man kann sie natürlich auch leicht abgewandelt einsetzen und das gesamte bzw. Teile des Computersystems herunterfahren, um gefahrloser das System wieder in einen redundanten Zustand zu bringen.

1.5.1 Hot-Plugging ohne Spare-Disk

Bei Hot-Plugging wechselt man die defekte Festplatte. Diese Methode **muss** vom Betriebssystem, SCSI-Controller, SCSI-Treiber und der Festplatte unterstützt werden. Hot-Plugging funktioniert nicht mit (E)(IDE) Festplatten.

Vorgehensweise

1. BS meldet den Ausfall einer HD, RAID-System läuft im Normalfall weiter
2. defekte HD mittels einer Software aus dem RAID-Array lösen, das RAID-System läuft normal weiter
3. defekte HD physisch auswechseln
4. neue HD evtl. partitionieren und ins RAID-Array einbinden
5. neue HD synchronisieren
6. Nun hat man wieder ein voll funktionsfähiges RAID-System

1.5.2 Hot-Plugging mit Spare-Disk

Man hat im Rechner eine Zusatzfestplatte welche automatisch die defekte HD ersetzt. Diese HD muss natürlich entsprechend vorbereitet sein (z.B.: partitioniert). Das RAID-System synchronisiert sich dann selbstständig und ist schnell wieder redundant. Das System meldet auch hier den Ausfall der HD und diese muss dann ausgetauscht werden.

1.6 Hardware-RAID

Beim Hardware-RAID wird das Zusammenspiel der Festplatten mittels spezielle Hardware organisiert, dem RAID-Controller.

Der RAID-Controller ist in physischer Nähe der Festplatten und kann in einem separaten Gehäuse, Storage Array, mit den Festplatten untergebracht sein. Bei den günstigen RAID-Controllern im Consumer-Bereich so günstig wie möglich zu bauen, überlässt man oft die RAID-Logik dem Prozessor. Diese Lösungen sind oft noch schlechter als reines Software-RAID

1.7 Software-RAID

Hier wird das Zusammenwirken der Festplatten komplett softwareseitig realisiert. Es ist keine besondere Hardware nötig, außer die entsprechenden Festplatten-Controller. Manchmal spricht man auch von Host based RAID, weil nicht das Speicher Subsystem, sondern der eigentliche Computer die RAID- Verwaltung durchführt. Die meisten modernen Betriebssysteme, wie Apple Mac OS X, IBM AIX, Linux, ab Microsoft Windows 2000 oder SUN Solaris sind dazu in der Lage.

Die einzelnen Festplatten sind in diesem Fall entweder über einfache Festplatten-Controller, Speicher-Controller wie RAID-Controller oder andere Subsysteme angeschlossen. Die Verwaltung der einzelnen Devices übernimmt der Computer. Der Vorteil von Software-RAID ist, dass kein spezieller RAID-Controller benötigt wird. Die vom Betriebssystem mitgelieferte RAID- Software oder eine separat installierte Software wird dann genutzt. Allerdings kann die Abhängigkeit von einem bestimmten Betriebssystem bzw. einer bestimmten Plattform auch ein Nachteil sein, und nicht zuletzt wird der Hauptprozessor des Computers bei Festplattenzugriffen belastet. Ebenfalls fällt die Möglichkeit weg, einen im RAID-Controller oder Speicher Subsystem vorhandenen Cache zu nutzen. In bestimmten Fällen kann Software-RAID schneller sein, z.B. wenn der *Volume Manager* des Betriebssystems die Zugriffe auf die Platten besser optimieren kann als der RAID-Controller. Das Betriebssystem kann den verfügbaren Hauptspeicher als Cache verwenden. Das Programm, welches die Verwaltung der Festplatten übernimmt wird gemeinhin als *Volume Management Software* oder *Volume Manager* bezeichnet.

1.8 Nutzen von RAID

In der heutigen Zeit sind die Rechner so schnell, dass der begrenzende Faktor fast immer die Festplatte ist. Das bemerkt man absonderst bei festplattenintensiven Anwenden, wie KDE oder StarOffice/OpenOffice. Durch den Einsatz von RAID fühlt sich das gesamte System deutlich schneller an, außerdem erreicht man eine höhere Datensicherheit. Das macht sich bei dem */home*-Verzeichnis bzw. *\Eigene Dateien*-Ordner positiv bemerkbar. Man ist besser vor Datenverlust geschützt.

RAID verursacht auch keine wesentlich höheren Kosten, denn dieses Feature wird bei vielen BS kostenlos mitgeliefert. Warum soll man dieses Feature nicht nutzen? Wer denkt, dass die CPU-Leistung stark abnimmt, den kann man beruhigen: Software-RAID 0 läuft schon einigermaßen akzeptabel auf einen 486DX-2 mit 66MHz und SCSI-Controller. Für die anderen RAID-Level und IDE-Platten sollte man schon eine CPU mit 200-300MHz haben. Die zusätzlichen Kosten, die durch die Anschaffung von einer höheren Anzahl von HD's anfallen, werden durch die fallenden Festplattenpreise kompensiert.

Benchmarks

Modus	# HD	Chunk-Size	Blockgröße ext2	schreiben	lesen
normal	1	-	4kByte	$s := Ref.$	$r := Ref.$
RAID 0	3	4kByte	4kByte	$2,6 * s$	$2,8 * r$
RAID 5	3	32kByte	4kByte	$1,7 * s$	$1,9 * r$
RAID 10	4	4kByte	4kByte	$1,7 * s$	$2,5 * r$

Die Daten sind nicht absolut zu sehen. Sie sollen nur einen Eindruck verschaffen welche Leistungssteigerungen bei RAID zu erwarten sind. Die s und r bezeichnen die Zeit, welche die HD für einen normalen Lese- oder Schreibvorgang benötigt. Bei beiden Vorgängen wurde vorausgesetzt, dass die Daten sequenziell zugegriffen wird.

Kapitel 2

Dateiverwaltung in Multiuser-Systemen

2.1 Einstieg

Die Dateiverwaltung ist ein wichtiger Bestandteil von modernen (Multiuser-) Betriebssystemen. Fast jede Anwendung, die auf einem Rechner läuft arbeitet mit Dateien bzw. deren Inhalt. Um die Dateien zu verwalten ist ein entsprechendes Dateisystem nötig, welches vom Betriebssystem bereitgestellt werden muss.

Ein Dateisystem in einer Multiuser-Umgebung sollte folgende Aufgaben lösen:

1. Dateien lesen, schreiben, löschen, ändern und erstellen;
2. Dateien vor den Zugriff anderer Benutzer schützen;
3. die Zugriffsrechte für seine Dateien selbst festlegen;
4. Dateien sicher und wiederherstellen;
5. Dateien selbst organisieren;
6. Dateien über symbolische Namen identifizieren;
7. Ressource Festplattenplatz für jeden Benutzer reglementieren;
8. das Dateisystem soll ökonomisch mit den vorhandenen Ressourcen* umgehen können.

*z.B. freier Platz, Rechenleistung, RAM-Benutzung

2.2 Arten von Dateisystemen

2.2.1 Lineare Dateisysteme

Die historisch ersten Dateisysteme waren lineare Dateisysteme auf Lochband oder Lochkarte sowie die noch heute für die Sicherung von Daten eingesetzten Magnetbandsysteme.

2.2.2 Hierarchische Dateisysteme

Frühe Dateisysteme hatten nur ein einzelnes Verzeichnis, das dann Verweise auf alle Dateien des Massenspeichers enthielt. In den meisten modernen Dateisystemen ist dieses Verzeichnis das Wurzelverzeichnis. Hier können Verzeichnisse neben normalen Dateien auch Verweise auf weitere Verzeichnisse, die Unterverzeichnisse, enthalten. Auch diese dürfen wieder Unterverzeichnisse haben.

Diese Verzeichnisstruktur wird auch als Verzeichnisbaum bezeichnet, mit einer Wurzel, dem Stammverzeichnis. Bei Windows ist es in der Regel `C:\`, bei UNIX/LINUX ist es ein `'/'` (*Slash*). In diesem Stammverzeichnis sind alle Dateien und Verzeichnisse samt ihrer Unterverzeichnisse eingehangen (*gemounted*).

Beispiele wie man auf die Datei *Vortrag.tex* zugreifen kann:

- `C:\Dokumente und Einstellungen\Michael\Eigene Dateien\BS Seminar\Vortrag.tex` (Windows 2000/XP)
- `/Users/Michael/BS Seminar/Vortrag.tex` (Mac OS X, (: statt / bei alten Versionen))
- `/home/Michael/BS Seminar/Vortrag.tex` (Unix / Linux)

Häufig bezeichnet der Begriff Dateisystem nicht nur die Struktur und die Art, wie die Daten auf einem Datenträger organisiert werden, sondern allgemein den ganzen Baum mit mehreren verschiedenen Dateisystemen (Festplatte(ext2, ext3, ReiserFS,...), CD-ROM, ...). Das merkt man besonders bei LINUX/UNIX da hier alle Geräte in das Stammverzeichnis gemounted werden. Bei Windows hingegen ist eher die Abgrenzung der verschiedenen Dateisysteme durch die Laufwerksbuchstaben gegeben. Korrekterweise müsste man hier von einem Namensraum sprechen, der von verschiedenen Teilnamensräumen (die eingebundenen Datenträgern mit deren Dateisystemen) gebildet wird, da aber dieser Namensraum sehr dateibezogen ist, wird häufig nur vom Dateisystem gesprochen.

2.2.3 Netzwerkdateisysteme

Die Systemaufrufe wie `open`, `read`, usw. können auch über ein Netzwerk an einen Server übertragen werden. Dieser führt dann die Zugriffe auf seine Festplatten durch und liefert die angeforderte Information an den Client zurück.

Da dieselben Systemaufrufe verwendet werden, unterscheiden sich die Zugriffe aus Programm- und Anwendersicht nicht von der auf die lokalen Geräte. Man spricht hier von transparenten Zugriffen, weil der Anwender die Umlenkung auf den anderen Rechner nicht sieht, sondern scheinbar unmittelbar auf die HD des entfernten Rechners schaut. Für Netzwerkdateisysteme stehen spezielle Netzwerkprotokolle zur Verfügung.

Kann auf ein Dateisystem z.B. in einem SAN von mehreren Systemen parallel direkt zugegriffen werden, spricht man von einem Globalen- oder Cluster-Dateisystem. Dabei sind zusätzliche Maßnahmen zu ergreifen, um Datenkorruption durch gegenseitiges Überschreiben zu vermeiden. Dazu wird ein Metadaten-Server eingesetzt. Alle Systeme leiten die Metadaten-Zugriffe - typischerweise über ein LAN - an den Metadaten-Server weiter, der diese Operationen wie Verzeichnisszugriffe und Block- bzw. Clusterzuweisungen vornimmt. Der eigentliche Datenzugriff erfolgt dann über das SAN als ob das Dateisystem lokal angeschlossen wäre. Da der Overhead durch die Übertragung an den Metadaten-Server insbesondere bei großen Dateien kaum ins Gewicht fällt, kann so eine Übertragungsgeschwindigkeit ähnlich der eines direkt angeschlossenen Dateisystems realisiert werden.

2.2.4 Datenbank-Dateisysteme

Neue Konzepte für Dateiverwaltung sind Datenbank-basierende Dateisysteme. Statt in einer hierarchisch aufgebauten Verwaltung, werden Dateien anhand ihrer Eigenschaften, wie Dateityp, Thema, Autor oder ähnlichen Meta-Informationen identifiziert. Die Formulierung einer Dateisuche kann man daher in SQL oder in natürlicher Sprache erfolgen.

Ansätze dafür sind GNOME Storage, WinFS und BFS. Dabei ist BFS bereits fertig entwickelt und bietet entsprechende Features einer Datenbank. Es wird in den Systemen BeOS und ZETA als Standard-Dateisystem genutzt.

Eine solche Form der Speicherung anhand von Meta-Informationen ist immer von der Kooperation der Nutzer abhängig, die das System mit geeigneten Meta-Informationen versorgen.

2.2.5 Spezielle virtuelle Dateisysteme

Das *open-read*-Modell lässt sich auch auf Geräte und Objekte anwenden, die normalerweise nicht über Dateisysteme angesprochen werden. Dadurch wird der Zugriff auf diese Objekte identisch mit dem Zugriff auf normale Dateien, was meist Vorteile bringt.

Unter den Linux-Kernels lassen sich System- und Prozessinformationen über das virtuelle *proc*-Dateisystem abfragen und ändern. Die virtuelle Datei */proc/cpuinfo* liefert z.B. Informationen über den Prozessor.

Pseudo-Dateisysteme wie *proc* gibt es einige unter Linux: *sysfs*, *usbdevfs* oder *devpts*; unter einigen BSD-Versionen gibt *kernfs*. Außerdem *devfs* bzw. *udev* für Gerätedateien. Diese Dateisysteme enthalten nur rein virtuell vorhandene Dateien mit Informationen oder Geräten, die auf eine „Datei“ abgebildet werden.

Der Kernel simuliert hier quasi die Existenz einer Datei, wie sie auch auf einer Festplatte vorhanden sein könnte. Dateien in *ramfs* oder *tmpfs*, aber auch *initrd* und ähnlichen Dateisystemen existieren jedoch tatsächlich, werden aber nur im Speicher gehalten. Sie werden aus Geschwindigkeitsgründen und aus logisch-technischen Gründen während der Boot-Phase eingesetzt.

2.3 Dateioorganisation

Festplatten haben normalerweise eine Blockstruktur, d.h. aus Sicht des Betriebssystems lassen sich Daten nur als ganze Blöcke lesen oder schreiben. Die Hardware der Festplatten präsentiert sich gegenüber dem Betriebssystem lediglich als große Fläche mit vielen nummerierten Blöcken. Ein Block umfasst meistens 512 (2^9) Bytes.

2.3.1 Dateiallokation

Wir wissen nun, das man die Festplatte in Blöcke unterteilt. Nun stellt sich die Frage wie man freie Blöcke eine neuen Datei zuordnet (*Dateiallokation*). Dabei sind folgende Dinge zu berücksichtigen:

1. Wenn man eine Datei erstellt, soll man ihr dann den maximal erforderlichen Platz zuweisen oder soll man Speicherplatz dynamisch zuweisen?
2. Wenn eine Datei über mehrere Blöcke geht, wie groß soll ein Block sein oder sollte man besser variable Blöcke nehmen oder soll man Eindatei-Blöcke nutzen?
3. Welche Struktur soll man für die Verwaltung der Blöcke nehmen?

Vergleich: Voraballokation und dynamische Allokation

Bei der Voraballokation muss die maximale Größe der Datei im Voraus bekannt sein. In einigen Fällen kann diese Frage beantworten, z.B.: Übersetzten von Programm, archivieren, Aber in den meisten Fällen ist eine Aussage schwer möglich oder gar unmöglich. Auf der einen Seite ist die Datei **nie** über die gesamte Festplatte verteilt, aber man verschwendet sehr viel Speicher, da man gezwungen ist immer mehr Speicher anzufordern als man braucht. Bei der dynamischen Allokation wird der Speicher in kleinen Stücken zugeteilt. Dabei wird der zur Verfügung stehende Speicher gut ausgenutzt, aber eine Datei kann über die gesamte Festplatte verteilt sein.

Blockgröße

Bei der Blockgröße gibt es zwei Extreme: Man kann die Blöcke so groß wählen, dass die gesamte Datei hinein passt oder es passt nur ein Bit, Byte oder Wort hinein. Man muss eine Blockgröße wählen, die einen gesunden Kompromiss zwischen Leistung auf das gesamte System und Leistung auf die einzelne Datei bezogen aufweist.

- zusammenhängender Speicherplatz ist gut in Bezug auf I/O-Operationen von großen Dateien
- viele kleine Blöcke lässt die FAT schnell wachsen
- Blöcke fester Größe sind bei der Neuallokation einfach zu handhaben
- bei Blöcken variabler Größe wird kein Speicherplatz verschwendet

Aus den Punkten lassen sich folgende Lösungen ableiten:

1. **Große Blöcke, variabler Größe** ermöglichen ein gutes Leistungsverhalten und es wird kein Speicherplatz verschwendet, außerdem ist die FAT klein. Es ist aber schwer frei werdenden Speicherplatz wieder zu verwenden.
2. Kleine **Blöcke** fester Größe bieten ein hohes Maß an Flexibilität. Jedoch wird die FAT größer, oder man muss sich eine gute Datenstruktur überlegen. Die Blöcke werden nach Bedarf zugeordnet.

Beide Lösungen gehen mit Voraballokation und dynamischer Allokation. Bei 1 und Voraballokation könnte man sogar die FAT einsparen. In der Praxis ist es aber vorteilhaft eine FAT zu haben, diese hat dann nur einen Eintrag pro Datei. Bei den Blöcken variabler Größe muss man die in jeden Fall, nach dem löschen, entstehende Fragmentierung[†] zu berücksichtigen. Die Lösungsansätze sind die gleichen wie beim Hauptspeicher. Es muss auch noch gesagt werden, dass es in modernen Dateisystemen wie *ext3* es erst gar nicht zu einer sehr hohen Fragmentierung kommt, solange die Festplatte noch ca. $\frac{1}{3}$ freien Platz aufweist.

Dateiallokationsverfahren

Im Laufe der Zeit haben sich drei Allokationsverfahren herausgebildet: zusammenhängende, verkettete und indizierte Allokation.

[†]Datei ist über einen nicht zusammenhängenden Bereich der Festplatte verteilt und es gibt freie Blöcke zwischen belegten Blöcken

	Zusammenhängend	Verkettet	Indiziert
Vorallokation?	erforderlich	möglich	möglich
Blockgröße	groß	klein	klein
Allokations- häufigkeit	einmal	selten bis häufig	häufig
Größe der FAT	ein Eintrag	ein Eintrag	umfangreich

Bei der **zusammenhängende Allokation** wird die Datei in zusammenhängenden Blöcken gespeichert. Wenn diese variable Größe haben entsteht keine externe und interne Fragmentierung, solange man keine Dateien löscht. Im Hinblick auf sequentiellen Dateizugriff ist die zusammenhängende Allokation das beste Verfahren. Man kann mehrere Blöcke gleichzeitig laden und man weiß immer an welcher Stelle der x . Block der Datei steht. Wenn man auf die Idee kommt Dateien zu löschen, dann kommt es unweigerlich zur externen Fragmentierung und nach einiger Zeit ist es nicht mehr möglich ausreichend große zusammenhängende Blöcke zu finden um eine weitere Datei anzulegen. Das hat zur Folge, dass man eine Speicherverdichtung durchführen muss. Dabei werden alle belegten Blöcke an den Anfang der Festplatte geschoben.

Wenn man die **verkettete Allokation** anwendet entsteht keine externe Fragmentierung. Denn jeder Block enthält einen Zeiger auf dem nachfolgenden Block. Das hat aber den Nachteil, dass die Datei über die gesamte Festplatte verstreut sein kann. Das kann man aber mit einer Datenzusammenlegung beheben. Dabei werden alle Blöcke einer Datei hintereinander gespeichert. Viel schlechter ist der Fakt, wenn man auf den x . Block zugreifen möchte, dann muss man erst alle $x - 1$ Blöcke davor lesen um zum x . Block zu gelangen. Die verkettete Allokation entspricht einer verketteten Liste. Wie bei einer verketteten Liste gibt es hier einen Startblock, dieser gibt die Länge der Datei in Blöcken an und hat den Zeiger zum ersten Datenblock.

Mit der **indizierten Allokation** werden die Nachteile der verketteten Allokation und der zusammenhängenden Allokation zum Teil behoben. Hier hat man eine mehrstufige Beschreibung wo die Datei gespeichert ist. Im einfachsten Fall enthält die FAT einen Eintrag, wo sich ein Index-Block befindet. In diesem stehen alle zur Datei gehörigen Blöcke. Damit das Lokalitätsprinzip besser funktioniert adressiert man Cluster. Man muss auch hier ab und zu eine Datenzusammenlegung durchführen, damit das System immer eine hohe Leistung hat. Der Vorteil ist, dass man sequenziell und direkt auf Block x seiner Datei zugreifen kann. Die indizierte Allokation ist die verbreitetste Form der Dateiallokation.

2.3.2 Cluster

Moderne Betriebssysteme fassen aus Performance- und Verwaltungsgründen mehrere Blöcke zu einem Cluster fester Größe zusammen. Heute sind Cluster mit acht

oder noch mehr Blöcken üblich, also 4096 Bytes pro Cluster. Die Clustergröße ist im Allgemeinen eine Zweierpotenz (2048, 4096 usw.)

So betrachtet ist eine Datei eine Speicherfläche beliebiger Größe, die auf der Festplatte aus einem oder mehreren Clustern besteht. Jede Datei erhält außerdem eine Beschreibungsstruktur, die die tatsächliche Größe, Referenzen auf die verwendeten Cluster und evtl. weitere Informationen wie Dateityp, Eigentümer, Zugriffsrechte enthalten kann.

2.4 Dateiverzeichnisse

Damit das Dateisystem weiß, wo welche Dateien sind und welche Attribute sie hat gibt es spezielle Dateien[‡] im Dateisystem wo diese Informationen gespeichert sind. Wie diese Informationen gespeichert sind ist sehr unterschiedlich. Auf diese Dateien hat der Benutzer keinen direkten Zugriff, nur über Dienste und Programme des Betriebssystems.

Gespeicherte Informationen

	Basisinformationen
Dateiname	Name der Datei, muss innerhalb des Verzeichnisses eindeutig sein
Dateityp	z.B.: Textdatei, ausführbare Datei usw.
Organisation	für Systeme, die mehrere Dateiorganisationen unterstützen
	Adressinformationen
Datenträger	Gerät auf dem die Datei gespeichert ist
Startadresse	phys. Startadresse auf der HD (z.B.: Zylinder-, Spur-, Blocknummer)
genutzte Größe	aktuelle Größe der Datei in Byte, Worte, Blöcke oder Cluster
zugeteilte Größe	maximale Größe der Datei, ohne das man neuen Speicher anfordern muss
	Zugriffskontrollinformationen
Besitzer	hat Kontrolle über Datei und kann Zugriffsrechte setzen
Zugriffsrechte	Wer darf mit der Datei was tun
	Nutzungsinformationen
Erstellungsdatum	Wann wurde die Datei zum ersten mal erstellt?
Ersteller	Ist in der Regel der Besitzer aber nicht zwingend
letzter Lesezugriff	Wann wurde die Datei zum letzten mal gelesen?
letzter Leser	Wer hat die Datei zuletzt gelesen?
letzte Änderung	Wann wurde die Datei zuletzt geändert?
Wer änderte	Wer hat die Änderung durchgeführt?
Sicherungskopie	Wann wurde die letzte Sicherungskopie auf ein anderes Medium gemacht?
Nutzung	aktuelle Aktivität der Datei: Prozesse, die sie nutzen; Ob sie im Hauptspeicher aktualisiert worden ist und auf der HD noch nicht

[‡]Es kann auch nur eine Datei sein

2.5 Zugriffsrechte

In einer Multiuser-Umgebung ist es absolut unerlässlich, das man Dateien mit Attributen versieht, welche regeln wer alles auf eine Datei zugreifen darf. Man muss auch sicherstellen, das bestimmte Bereiche des Systems für den normalen Benutzer unsichtbar sind, d.h. er soll noch nicht einmal wissen, das es bestimmte Dateien gibt. Jeder einzelne Benutzer muss die Freiheit haben, welche Zugriffsrechte er welchen Dateien gibt. Beispiele für solche Rechte:

- **Keine** Der Benutzer erfährt nicht von der Existenz einer Datei, das betreffende Verzeichnis darf man in der Regel nicht öffnen.
- **Kenntnis** Man erfährt, dass eine bestimmte Datei existiert. Man kann die Attribute[§] der Datei lesen aber man darf sie nicht öffnen oder gar verändern.
- **Ausführen** Man kann ein Programm ausführen oder ein Verzeichnis öffnen. Man kann das Programm aber nicht kopieren.
- **Lesen** Man kann den Inhalt einer Datei lesen. Es gibt aber Implementationen, die unterscheiden zwischen lesen und kopieren. In den meisten Systemen kann man auch eine Datei kopieren, wenn man sie lesen darf.
- **Anhängen** Man darf die Datei durch Anfügen von Daten vergrößern. Es ist aber untersagt Daten zu löschen und zu verändern.
- **Aktualisieren** Man kann den Inhalt einer Datei beliebig verändern. Die Datei darf man aber nicht löschen. Es gibt auch Systeme, die zwischen den unterschiedlichen Graden von aktualisieren unterscheiden, so ist es evtl. gut wenn man nur Daten modifizieren darf, d.h. ich ersetze Inhalte der Datei durch Neue.
- **Schutz ändern** Dieses Privileg hat in den meisten Systemen der Besitzer der Datei.
- **Löschen** Man kann die Datei aus dem Dateisystem entfernen

Die Liste der Berechtigungen ist **nicht** als Hierarchie zu sehen. So kann es sinnvoll für Protokoll-Dateien sein, dass niemand, außer der Administrator, die Dateien lesen kann. Damit unterbindet man einige Angriffsarten auf das eigene System. Mit einer Hierarchie der Rechte lässt sich auch keine Briefkasten-Funktion auf Dateisystemebene realisieren, d.h. man darf in ein Verzeichnis schreiben, aber nicht lesen. Um ein wirklich gutes und flexibles Zugriffssystem zu haben **müssen** die einzelnen Rechte **unabhängig** voneinander zuteilbar sein.

[§]Benutzer, Größe, Zugriffsrechte, ...

Für eine Multiuser-Umgebung ist es auch unerlässlich, dass jeder Benutzer einer Gruppe zugeteilt ist. Das hat zur Folge, dass man bestimmte Rechte sich selber, einer Anzahl von Benutzern, der eigenen Gruppe, einer Anzahl von Gruppen und den Rest der Welt gewähren kann. Es ist durchaus sinnvoll, dass man selbst sich nicht alle Rechte gibt, z.B.: Schutz von Backups, wichtigen Daten,

2.6 Das Dateisystem ext3

Neben den schon beschriebenen Kennzeichen eines guten Dateisystems gibt es noch eine Hand voll andere Merkmale, die ein wirklich gutes Dateisystem ausmachen. Was ein gutes Dateisystem ist, soll hier exemplarisch an ext3 (*third extended filesystem*) gezeigt werden.

Bei den bisherigen Betrachtungen geht man davon aus, dass das System arbeitet immer einwandfrei. Wenn aber eine Operation auf dem Dateisystem fehl schlägt oder der Rechner abstürzt, kann im schlimmsten Fall das Dateisystem zerstört werden oder man benötigt sehr viel Zeit, um die Integrität der Daten zu prüfen. Einen Ausweg schafft hier ein *Journal*. Wichtig ist auch für Dateisysteme, die auf mehreren Architekturen eingesetzt werden, dass die Daten in einem einheitlichen Format geschrieben werden. Sonst kann es sein, dass die Leserichtung auf einmal nicht mehr stimmt. Bei ext3 ist es das Little Endian-Format für alle Informationen.

Journal

Wenn man eine Änderung am Dateisystem durchführt, wird sie als Transaktion im Journal vermerkt und sie kann dann im Fall eines Absturzes entweder abgeschlossen oder noch nicht abgeschlossen sein. Wenn eine Transaktion zum Absturzzeitpunkt (oder im Normalfall, wenn das System nicht abstürzt) abgeschlossen war, ist garantiert, dass alle an dieser Transaktion beteiligten Blöcke einen gültigen Dateisystemstatus repräsentieren. Diese Blöcke werden dann ins Dateisystem kopiert. Wenn eine Transaktion zum Absturzzeitpunkt nicht abgeschlossen war, kann nicht garantiert werden, dass die beteiligten Blöcke konsistent sind, daher wird eine solche Transaktion verworfen (das bedeutet, dass die Dateisystemänderung, die diese Transaktion repräsentierte, verloren geht).

Bei abgebrochenen Schreiboperationen kann es passieren, dass ein Teil einer Datei bereits aus den neuen Daten besteht und ein Teil noch aus den alten, was manchmal noch schlimmer sein kann als ein inkonsistentes Dateisystem. ext3 schützt nicht davor, dass Daten verloren gehen, die zum Absturzzeitpunkt zwar eigentlich bereits auf die Platte geschrieben sein sollten. Das liegt daran, dass die Daten erst vom Kernel gepuffert werden. Nach dem Abspielen des Journals ist nur garantiert, dass man mit einem konsistenten Datenbestand zu einem gegebenen Zeitpunkt weiterarbeiten kann.

Bei ext3 gibt es 3 Journaling-Stufen: Full, Writeback und Ordered. Bei Full werden die Dateiinhalte und die Metadaten erst ins Journal geschrieben und dann ins Dateisystem. Also benötigt man zum Schreiben einer Datei die doppelte Zeit. Man erhält dafür maximale Sicherheit. Bei Writeback werden nur die Metadaten ins Journal geschrieben und die Daten werden direkt ins Dateisystem geschrieben. Wenn im Absturzfall sich eine Datei in einem Schreibvorgang befand, wird sie evtl. beschädigte Daten enthalten. Einen guten Mittelweg zwischen maximaler Sicherheit und hoher Leistung bietet der Ordered-Modus. Hier wird erzwungen, dass der Dateiinhalt erst nach den Metadaten ins Dateisystem geschrieben wird.

2.6.1 Reservierter Speicherplatz

Innerhalb des Dateisystems kann man eine bestimmte Anzahl von Clustern für einen bestimmten Benutzer reservieren, normalerweise für *root*. Dies erlaubt dem System, auch dann zu funktionieren, wenn nichtprivilegierte Benutzer den gesamten ihnen zur Verfügung stehenden Speicherplatz aufgefüllt haben. Der Mechanismus funktioniert unabhängig von Dateisystem *quotas*[¶]. Weiterhin hilft er dabei, ein vollständiges Füllen des Dateisystems zu verhindern und so Fragmentierung zu bekämpfen.

2.6.2 Datensicherheit

Manchmal reichen Zugriffsrechte nicht aus um ausreichende Sicherheit zu gewährleisten. Abhilfe schafft hier eine Verschlüsselung des Dateisystems. Eine Verschlüsselung des Dateisystems bietet ext3 nicht an. Aber man kann für den Bootvorgang nicht relevante Verzeichnisse über sogenannte *Loopback-Device* verschlüsseln. Ein Punkt der bei ext3 von vielen bemängelt wird, ist das ext3 beim löschen eine die Blockzeiger der Inodes mit Nullen überschreibt. Dadurch wird die Rekonstruktion gelöschter Daten relativ schwer.

[¶]quotas sind Begrenzungen in der Anzahl der Dateien oder Cluster, die ein Benutzer oder eine Gruppe belegen kann

Liste der Abkürzungen

Abk.	Beschreibung
HD	H ard D isk (<i>Festplatte</i>)
vgl.	vergleiche
engl.	englisch
I/O	I nput/ O utput (<i>Ein-/Ausgabe</i>)
Kap.	Kapazität
#	Anzahl, wenn in Formeln verwendet
BS	B etriebssystem
Ref.	Referenz
phys.	p hysikalisch
FAT	F ile A llocation T able (<i>Dateizuordnungstabelle</i>)