Journaling-Dateisysteme im Überblick

Block-Organisatoren

Unsichtbar und zuverlässig sollen die vier Journaling-Dateisysteme sein, die im Kernel 2.6 zur Wahl stehen. Ext 3, JFS, Reiser und XFS übertreffen Ext 2 in puncto Performance und fragmentieren weniger schnell. Dank des Journals sind abgestürzte Rechner zudem schneller wieder einsatzbereit. Jörg Reitter

Das Dateisystem legt sich wie eine Schicht aus lauter winzigen Blöcken auf die Oberfläche der Platte. Anfangs ist diese Schicht geordnet, denn alle Daten liegen schön sauber hintereinander in zusammengehörigen Blöcken. Sobald jedoch die Schreibvorgänge beginnen, geht die Ordnung verloren. Dabei tritt der Effekt der externen Fragmentierung auf. Moderne Linux-Dateisysteme kommen mit dem Alterungsprozess besser zurecht als das in die Jahre gekommene Ext-2-Dateisystem. So hat ein ausführlicher Vergleich zwischen Ext 2 und ReiserFS [1] ergeben, dass ReiserFS deutlich geschickter auf die Leistungsbremse Fragmentierung reagiert.

Ausgedient haben ausgedehnte Kaffeepausen, wie sie fällig werden, wenn das Analysewerkzeug Fsck ein Ext-2-Dateisystem prüft. Neuere Linux-Dateisysteme protokollieren Änderungen in einem Katalog, der den Datenfundamenten ihren Namen gibt: Journaling-Dateisysteme. Dank des Journals hat das Fsck-Programm Inkonsistenzen bereits repariert, wenn »Fsck.ext2« noch untersucht, ob es überhaupt Fehler beseitigen muss. Grund: »Fsck.journal« prüft nur jene Daten, die älter sind als der letzte Eintrag im Journal. Der Rechner ist schneller einsatzbereit und die Verfügbarkeit verbessert sich.

Auf die Größe kommt es an

Wichtig ist die hohe Verfügbarkeit ohnehin, da der Datenberg unaufhörlich wächst und mit ihm auch die Kapazität der Hardware. Die Festplattenhersteller haben die 300-GByte-Grenze überschritten und Raid-Systeme verwalten schon längst mehrere Terabyte. Dateisystemdesigner denken allerdings weit über die heute verfügbare Hardware hinaus und schaffen Systeme, die mehr als eine Million Terabyte effizient verwalten können (siehe Tabelle 1). XFS von SGI führt bei der theoretisch möglichen Dateisystemgröße knapp mit 1,125 EByte vor ReiserFS, das bei 1 EByte an seine Grenze stößt. IBM folgt als Dritter mit 32 PByte. Ext 3 hält sich an die Leistungsgrenze des Kernels 2.6, der die Dateisystemgröße auf 16 TByte beschränkt.

Geht der Speicherplatz zur Neige, vergrößert der Benutzer das Dateisystem im laufenden Betrieb. Sinnvoll ist dies vor allem im Zusammenhang mit einer logischen Festplatte (LVM) oder einem Software-Raid. Mit beiden Techniken beschäftigt sich der Artikel „Datenkoppler“ in dieser Ausgabe.

Dateisysteme im Kernel 2.6

Für jedes der Dateisysteme Ext 3, JFS, ReiserFS und XFS befinden sich Treiber im Kernel 2.6, wodurch der Anwender nun bequem testen kann, ohne die Originalquellen zu patchen. Neben den blanken Treibern enthält der Kernelbaum auch fortschrittliche Funktionen wie Quotas, Access Control Lists (ACL) und Unterstützung für Debugging (Abbildung 1). Gleich bleibt, dass das Dateisystem der Root-Partition in den Kernel kompiliert sein muss, falls der Admin keine Initial-RAM-Disk verwendet. Wie gewohnt kann der Benutzer die Dateisystemtreiber auch als Modul übersetzen und zur Laufzeit laden.

Was die Verwaltung von Dateisystemrechten angeht, riefen die Netzwerkadministratoren in letzter Zeit immer deutlicher nach ACLs. Diese erweitern das übliche Rechteschema Eigentümer, Eigentümergruppe und andere, sodass der Admin beispielsweise entfernten Win-

dows-Benutzern einfach Zugriff auf Dateien auf einem Samba-Server geben kann. Für den Windows-Benutzer erscheinen die Dateirechte so, als greife er statt auf Samba auf einen Windows-NT/2000-Server zu. Mehr zur Konfiguration bei ACLs ist auf den Websites [2] und [3] zu finden.

Aufteilen und sichern

Auf Fileservern gibt es ohnehin so gut wie immer Probleme mit dem Platz. Quotas helfen wirksam, dass der Server nicht voll läuft. Jedem Benutzer weist der Administrator ein Stück des Dateisystems zu. Ext 2/3 arbeitet mit den Standard-Quotas im Kernel 2.6 zusammen und XFS bringt eigene Unterstützung mit. Die Entwicklung von JFS und ReiserFS ist hingegen noch nicht so weit. Bei IBM steht der Quota-Support auf der Todo-Liste. ReiserFS wird die unverzichtbare Hilfe erst ab Version 4 enthalten. Wer ReiserFS mit Quotas benötigt, muss einen Kernel 2.4 patchen.

Bei den Utility-Programmen sieht es ähnlich aus: JFS und ReiserFS bringen keine Dump- und Restore-Programme mit. Immerhin enthält Suses Yast 2 eine Backup-Schnittstelle für ReiserFS. Allerdings sind damit keine inkrementellen Datensicherungen möglich. Anwender müssen sich daher zwangsweise an Tar oder Cpio gewöhnen. User von Ext 3 dagegen benutzen einfach die Programme, die sie von Ext 2 gewohnt sind [4]. Die Open-Source-Abteilung bei SGI ist ebenfalls schon weiter und packt eigene Backup-Helfer in das Paket Xfsprogs [5]. Natürlich gibt es auch andere Software, die inkrementelle Backups anfertigen kann, in einem Unternehmen sollte dafür auch ein entsprechendes Budget bereitstehen [6].

Mit Journaling schneller

Dateisysteme, die ein Journal führen, tragen einen großen Teil zu hoher Verfügbarkeit bei. So ausgestattete Systeme sind nach einem Absturz viel schneller wieder einsatzbereit als jene, die auf ein Dateisystem ohne Journal wie Ext 2 bauen. Ein ungeplanter Reboot birgt das Risiko einer Dateisystem-Inkonsistenz in sich, da Dateien, die gerade in Bearbeitung waren, nicht sauber geschlossen wurden. Beim Neustart muss das »fsck .ext2«-Programm alle Dateien prüfen, was bei großen Dateisystemen Stunden dauern kann. Nicht so, wenn ein Journaling-Dateisystem als Fundament dient. »Fsck.journal« prüft nur die Daten, die älter sind als der letzte Eintrag im Journal. Das System ist in Sekunden statt in Stunden wieder einsatzbereit.

Aber nicht nur nach einem Absturz sind die Journaling-Dateisysteme überlegen. Auch was die organisatorische Effizienz auf unterster Ebene angeht, schlagen sie Ext 2 um Längen. Als Organisationsschema setzen die Journaling-Systeme auf bestimmte Such- und Sortier-Algorithmen, die so genannten B-Trees (siehe Kasten „Balancierte Bäume“).

Im Gegensatz zu einer ungeordneten Liste wie bei Ext 2 sind Verweise auf die

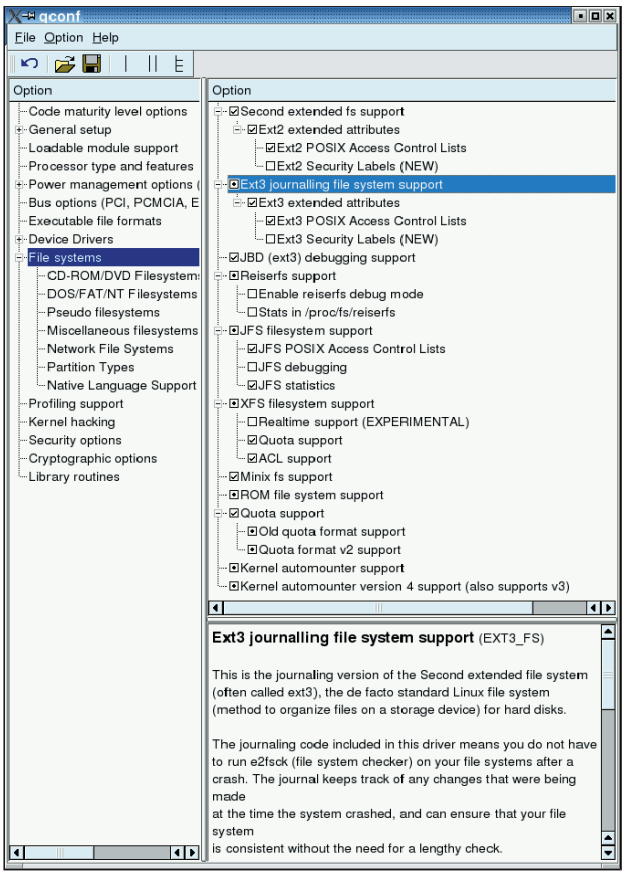

Abbildung 1: Ext 2/3, JFS, ReiserFS und XFS haben den Sprung in den Kernel 2.6 geschafft. Unterschiede sind beim ACL-Support sowie bei Quotas und Debug-Funktionen zu verzeichnen.

Balancierte Bäume

Mit Balanced Trees (B-Trees, ausgeglichene Bäume) folgen die Linux-Dateisysteme einem Konzept, das bereits seit über 30 Jahren auf Datenbanken angewendet wird. Durch die regelmäßige (ausgeglichene) Verteilung der Daten bei Journaling-Dateisystemen entsteht ein Organisationsschema, das effizienter als das Verfahren des Ext-2-Dateisystems arbeitet. Ext 2 arbeitet mit ungeordneten Listen, die ineffizient beim Suchen und Löschen sind, da im ungünstigsten Fall das gesamte Verzeichnis nach der Datei durchsucht werden muss.

Mit B-Trees kommen Verzweigungen in die Organisationsstruktur. Verzweigungen, die von der gleichen Wurzel ausgehen, sind immer gleich tief, sodass die Ausgeglichenheit gewahrt bleibt. Damit erhöht sich die Effizienz beim Durchsuchen großer Datenbestände, da niemals ein ganzes Verzeichnis durchsucht werden muss. Die Daten sind in einer baumartigen Struktur organisiert. Die Blätter des Baums speichern die Daten, die Verzweigungen (Knoten) enthalten Verweise auf die nächste Baumebene oder zuletzt die Blätter.

Ausgefeilte Algorithmen sorgen für maximale Geschwindigkeit

Der Standard-B-Tree speichert Daten und Schlüssel in internen Nodes und Blatt-Nodes. Ein Node ist ein Element des Baums und kann Wurzel, Verzweigung oder Blatt bedeuten. Die maximale Anzahl von Knoten, die zum untersten Knoten (dem Blatt) führen, entspricht der Höhe eines Teilbaums (allen Elementen unterhalb eines Knotens). Der Baum ist balanciert, wenn sich die minimale Anzahl von Knoten auf dem Weg von der Wurzel eines jeden Teilbaums hin zu einem Blatt von der maximalen Anzahl auf einem beliebigen anderen Weg höchstens um eins unterscheidet. Eine ausführliche Erklärung von B-Trees finden Anwender in der Linux-Fibel [www.linuxfibel.de/filesys.htm#supp2]. Wie die Implementierung in C funktioniert, erklärt das Dokument „Sorting and searching algorithms“, das auf [http://epaperpress.com/sortsearch/download/sortsearch.pdf] bereitliegt.

einzelnen Blöcke, die eine Datei repräsentieren, in einem Baum organisiert. Die Struktur ist so ausgefeilt, dass ein Prozess höchstens halb so lange warten muss, bis das Dateisystem die zur Datei gehörigen Blöcke gefunden hat.

Da die Zweige von der Wurzel aus gesehen immer die gleiche Tiefe haben, löst sich der Begriff B-Trees in der Fachliteratur zu balancierten Bäumen auf. Jedes der vier Journaling-Dateisysteme benutzt eine andere Variante, was das eine Dateisystem effizient bei vielen kleinen Dateien in einem Verzeichnis macht und andere eher mit großen Dateien zurechtkommen lässt.

Ein Engpass, der alle Dateisysteme außer Ext 3 betrifft, ist das Virtual-Filesystem-Switch (VFS). Es ist auf Inode-basierende Dateisysteme ausgelegt. ReiserFS, JFS und XFS benutzen statt der Inodes jedoch B-Trees. Diese beschleunigen den Suchvorgang, haben aber auch das Problem, dass bereits ein einziger Fehler in der Datenstruktur zu einem Verlust von Daten führen kann. Zudem ist der Export des Dateisystems per NFS problematisch und mit den Standard-Quotas im Kernel kommen JFS, ReiserFS und XFS ebenfalls nicht zurecht.

Tabelle 1: Dateisystem-Features

Feature	Ext 3	JFS	ReiserFS	XFS
Journaling: Daten/Metadaten	✓/✓	-/✓	-/✓	-/✓
Journal-Replay Kernel/Userspace	✓/✓	-/✓	✓/-	✓/-
Journal intern/extern	✓/✓	✓/✓	✓/-	✓/✓
B-Trees-basierte Verzeichnisse	-	✓	✓	✓
Dynamische Inode-Allocation-Map	-	✓	✓	✓
Extended Attributes/Posix-ACLs	✓/✓	-/✓	-/-	✓/✓
Quotas	✓	-	-	✓
Offline Vergrößerung/Verkleinerung	✓/✓	✓/-	✓/✓	✓/-
Online Vergrößerung/Verkleinerung	-/-	-/-	✓/-	✓/-
Implementierte Blockgröße (in MByte)	1, 2, 4	4	4	4
Maximale Dateisystemgröße	16 TByte	32 PByte	1 EByte	1,125 EByte
Maximale Dateigröße	2 TByte	4 PByte	1 EByte	562,5 PByte

Ext 3 oder Journaling Ext 2

Wer auf Ext 3 [7] setzt oder noch Ext 2 bevorzugt, geht kein Risiko ein. Nicht nur, dass der Anwender ein Ext-2-System sehr einfach mit dem Programm Tune2fs zu einem Ext 3 machen kann, ist ein gewichtiger Vorteil. Auch an den Dump- und Restore-Befehlen für die Datensicherung ändert sich nichts. Ext 3 ist lediglich um die Journalfunktion erweitert, die ihrerseits noch die drei Optionen »journal«, »writeback« und »ordered« mitbringt.

Bei der Option »ordered« protokolliert das Journal die Metadaten. Ändern sich eine Datei und die dazu gehörigen Metadaten, werden diese Daten vorher gesichert (Flush). Dies ist das Standardverhalten. Im »writeback«-Modus führt Ext 3 lediglich die Metadaten im Journal und sichert die Daten nur im Zuge des »sync«-Aufrufs. Der zwar sicherste, aber auch der langsamste Modus ist »journal«, der Änderungen an Metadaten und Dateien protokolliert.

Größter Nachteil von Ext 3 ist die Verzeichnis-Implementierung. Statt B-Trees verlässt sich das System auf so genannte Linked-Lists. Bei großen und reichlich gefüllten Dateisystemen benötigt Ext 3 deutlich mehr Zeit, um angeforderte Dateien zu finden. Ebenso schwach verhält es sich bei der externen Fragmentierung. Eine Untersuchung, wie sich das Dateisystem unter Langzeitbedingungen bewährt, zeigt eine Diplomarbeit [1].

Im Vergleich zu den anderen Journaling-Dateisystemen bleibt Ext 3 leistungsmäßig auch bei Spezialfällen wie sehr großen Dateien (größer als 1 TByte) und Partitionen oder sehr vielen Dateien in einem Verzeichnis zurück. Die internen Strukturen weisen zu wenige Bits auf, um solche Fälle gut zu handhaben.

JFS noch zu mager

JFS [9] von IBM zielt auf Datenbank- und Fileserver sowie andere Anwendungen mit hohem Bedarf an Festplattenspeicher. Außerdem stehen bei IBMs Entwicklung die Datensicherheit und die Skalierbarkeit ganz oben. Allerdings hinkt JFS für Linux der Entwicklung ein wenig hinterher. Die Todo-Liste ist im Vergleich am längsten und das System bringt die wenigsten Optionen mit. Immerhin lässt sich mittlerweile eine JFS-formatierte Partition im laufenden Betrieb vergrößern. Posix-ACLs unterstützt JFS ebenfalls.

Problemen mit dem Dateisystem kommt der Anwender auf die Spur, wenn er das JFS-Debugging im Kernel-Setup aktiviert. Die dabei generierten Meldungen gehen an Syslog. Auch an die Statistiker unter den Administratoren hat IBM gedacht. Eine entsprechende Option bietet das Kernel-Setup. Die Statistik sieht der Benutzer Root in »/proc/fs/jfs/«.

Die Verzeichnisorganisation ist in JFS sehr geschickt gelöst. Kleine Verzeichnisse mit maximal acht Einträgen speichert das System direkt im Inode. Alle größeren Verzeichnisse sind in einem B+-Tree organisiert. Für riesige Dateien und Partitionen ist JFS als 64-Bit-Dateisystem zudem gut gerüstet. Die Userspace-Tools sind in dem Paket Jfsutils zusammengefasst. Für den 2.6er Kernel benötigen Anwender mindestens die Version 1.0.14.

ReiserFS mag kleine Dateien

Zwar heißt es seit geraumer Zeit, dass die vierte Version von Hans Reisers Dateisystem bald die FTP-Server dieser Welt beglücken werde. Es tut sich aber noch nichts, abgesehen von einer Ankündigung auf Reisers Firmenwebseite [8]. Daher nimmt dieser Artikel die Fähigkeiten des letzten Upgrades der Version 3 unter die Lupe, wie sie sich als ReiserFS 3.6 auch im aktuellen Kernel befindet. Wer die Release der Version 4 nicht verpassen möchte, lässt sich auf die recht aktive Mailingliste setze.

Die herausragende Funktion von ReiserFS ist die sehr gute Performance, wenn sich viele kleine Dateien im System befinden. Klein bedeutet, dass sie weniger als 1 KByte groß sind. Solche Dateien kommen häufig auf News-Ser-

vern vor, weshalb Admins das System auch gerne für diesen Netzwerkdienst empfehlen. Nicht so leistungsstark ist ReiserFS, wenn das System viele Dateien anlegt und löscht, beispielsweise im Verzeichnis »/tmp«.
ReiserFS geht sehr effizient mit Dateien um. Es allokiert nicht in festen 4-KByte-Blöcken, sondern in exakter Größe. Zudem komprimiert es das Ende mehrerer Dateien, die keinen ganzen Block füllen, und speichert diese zusammen in einem Block (Tail Packing). Bei der Skalierbarkeit ist ReiserFS gut gerüstet für die Zukunft. Theoretisch kann der Anwender ein 1 EByte großes Dateisystem anlegen und darin auch mit einer bis zu 1 EByte großen Datei arbeiten.

XFS schon sehr weit

XFS [10] offeriert deutlich mehr Drumherum als JFS oder ReiserFS. Es gibt eine Menge Optionen, die Dokumentation ist sehr gut und zu jedem Tool in den Xfsprogs existiert eine ausführliche Manpage. SGIs Open-Source-Gabe kann im Kernel 2.6 bereits mit ACLs aufwarten. Zudem offeriert es als einziges Dateisystem eine eigene Unterstützung für Quotas. XFS behandelt die Quota-Information als Metadaten und integriert sie in das Journal. Somit stellt es auch höchste Konsistenz der Quotas sicher. Nicht zuletzt ist das Datenformat der Quotas kompatibel mit dem Format von SGIs hauseigenem Irix-Betriebssystem und erlaubt so die Migration, ohne die Systeme zu konvertieren.
XFS ist wie JFS ein 64-Bit-Dateisystem und daher gut vorbereitet auf riesige Dateien. Theoretisch kann das Dateisystem 18 EByte groß sein und nur zwei Dateien mit 9 EByte aufnehmen. Nicht überraschend bietet es als einziges Journaling-FS eine Online-Vergrößerung an.

Leistungseinbruch – durch externe Fragmentierung programmiert

Eine Datei der Größe 10 KByte liegt auf einem Dateisystem mit einer Blockgröße von 4 KByte. Die Datei passt somit nicht in einen Block und das Dateisytem verteilt die Datei auf drei Blöcke. Diese liegen anfangs hintereinander. Nun ist in einem Dateisystem ziemlich viel los: Die Benutzer löschen Dateien oder fügen neue hinzu und Protokollprozesse vergrößern ihre Logs. Bei diesem regen Treiben bilden sich immer mehr Blockinseln, wo vorher ein solider Dateibrocken war. Die externe Fragmentierung erfolgt ganz kontinuierlich, was sich wie eine langsam anziehende Bremse bemerkbar macht (siehe auch [1]).
Ein Defragmentierprogramm wäre ideal, um die Ordnung wiederherzustellen. Allerdings ist die Defragmentierung bei den heutigen Festplattengrößen nur zu schaffen, wenn Zeit keine Rolle spielt. In einer produktiven Umgebung ist es daher besser, zu formatieren und die Datensicherung einzuspielen. Egal ob es sich um 200 GByte oder 20 TByte Daten handelt – irgendwann holt der Tod jedes Dateisystem ein.

Fazit

Journaling-Dateisysteme sind sich bis auf Ext 2/3 technisch recht ähnlich. Nur im Entwicklungsstand gibt es Unterschiede. Da sich Ext 2 leicht zu einem abwärts kompatiblen Ext 3 machen lässt, dürfte es für viele Anwender das Dateisystem der Wahl sein. Was Ext 3 an Leistungsfähigkeit fehlt, gleicht es mit Robustheit aus. Wer hingegen ein Dateisystem mit enormer Skalierbarkeit und hoher Performance sucht, kommt nicht an XFS vorbei. Viele Utilities und die gute Dokumentation sind weitere Pluspunkte für SGIs Dateisystem. ReiserFS 3.6 empfiehlt sich, wenn viele kleine Dateien im System liegen. Die JFS-Entwicklung ist nicht so weit wie die der anderen Dateisysteme. Für Produktivbereiche ist es daher noch nicht geeignet. ■

Infos

[1] Fragmentierung: [http://www.informatik.uni-frankfurt.de/~loizides/reiserfs/thesis.html]
[2] ACL-Kurs: [http://acl.bestbits.at]
[3] Suse-ACL: [http://portal.suse.de/sdb/de/2002/10/81_acl.html]
[4] Dump/Restore für Ext 2/3: [http://sourceforge.net/projects/dump/]
[5] Xfsprogs: [ftp://oss.sgi.com/projects/xfs/]
[6] Backup-Software: [http://www.linuxmafia.com/faq/Admin/tape-backup.html]
[7] Ext 3: [http://www.zip.com.au/~akpm/linux/ext3/]
[8] ReiserFS: [http://www.namesys.com]
[9] JFS: [http://oss.software.ibm.com/jfs/]
[10] XFS: [http://oss.sgi.com/projects/xfs/]