Neues Journaling-Dateisystem mit atomaren Transaktionen

IV. Symphonie von Reiser

Das neu entwickelte Reiser 4 steht kurz vor seiner Premiere im Kernel 2.6. Es verspricht atomare Transaktionen, schreibt und liest flotter als sein Vorgänger ReiserFS, nutzt die Platte besser und ist durch Plugins erweiterbar. Dancing Trees dirigieren dieses File-Orchester. Marcel Hilzinger

Nach über vier Jahren Entwicklung scheint das Reiser-Dateisystem dieses Jahr den Schritt zur Version 4 zu schaffen. Das Entwicklerteam um Hans Reiser programmierte Reiser 4 von Grund auf neu. Derzeit testet das Team die Performance und Stabilität des Nachfolgers von ReiserFS und bereinigt kleinere Fehler. Das neue Journaling-Dateisystem beherrscht unter anderem atomare Transaktionen und lässt sich durch Plugins erweitern. Auf der Namesys-Homepage [1] steht eine Testversion zum Download bereit (siehe Kasten „Installationsanleitung“).

Grundlegender Aufbau

Vereinfacht besteht das Reiser-4-Dateisystem aus zwei Schichten: der Speicherschicht (Storage Layer) und der semantischen Schicht (Semantic Layer). Auf beiden befinden sich Objekte. Reiser 4 unterscheidet nicht zwischen Dateien und Verzeichnissen, sondern behandelt beide gleichermaßen als Objekte. Jedes Objekt auf dem Dateisystem verfügt über eine Objekt-ID und einen Schlüssel (Key), der teilweise aus der Objekt-ID generiert wird.

Wie der Name vermuten lässt, sind Keys die Schlüssel zum Dateisystem. Reiser 4 findet jedes Objekt über seinen Schlüssel. Es ist sogar möglich, über den Schlüssel einen bestimmten Abschnitt einer Datei zu finden. In der Grundeinstellung nutzt Reiser 4 neuerdings lange Schlüssel (zum Beispiel »10001:1:746f6d 20776169:0:10278:0«), die aus folgenden Elementen bestehen:

- Major Locality
- Minor Locality
- Anordnung (Ordering)
- Unbenutzt (immer 0)
- Objekt-ID
- Offset

Während die Speicherschicht für die eigentliche Lagerung und das Anordnen der Daten in den Baumstrukturen zuständig ist, übernimmt die semantische Schicht das Auflösen von Dateinamen zu Schlüsseln. Mit Hilfe des Schlüssels findet das Filesystem dann in der Speicherschicht die gesuchten Bytes. Der Offset gibt die Byte-Position innerhalb eines Objekts an.

Plugins für das Filesystem

Reiser 4 verwirklicht Dateioperationen mit Hilfe von Plugins. Die grundlegenden Unix-Dateiverwaltungsarbeiten erledigt das Dateisystem-interne File- und Directory-Plugin. Jede Datei und jedes Verzeichnis besitzt zudem eine Plugin-ID, die eine Sammlung externer Methoden bezeichnet. Über diese Methoden lassen sich Dateisystem-externe Operationen mit dem File ausführen.

Dateisystem-externe Operationen sind Möglichkeiten zur Dateiverwaltung, die nicht im Filesystem integriert sind. Sie erlauben es externen Programmen, in die Interna des Dateisystems einzugreifen, ohne dazu einen neuen Kernel kompilieren zu müssen. Reiser 4 besitzt zurzeit über 30 interne Plugins, die der Befehl »fsck.Reiser 4 -l« listet. Externe Plugins sind noch nicht bekannt. Wünschenswert wäre unter anderem ein Quota-Plugin – diese Funktion unterstützt Reiser 4 noch nicht.

Das Anordnen der Objekte wird vom File- und vom Hash-Plugin übernommen. Im Gegensatz zu den meisten Linux-Dateisystemen ordnet Reiser 4 Dateien nicht nach dem Erstellungsdatum, sondern alphabetisch. Bei Dateien, deren Name länger als 15 Buchstaben ist, entscheiden zunächst die ersten acht Buchstaben, dann der Hashwert des gesamten Dateinamens.

Zum Anordnen der Verzeichnisse benutzt Reiser 4 das Hash-Plugin. Es benutzt derzeit Cookies, die aus dem

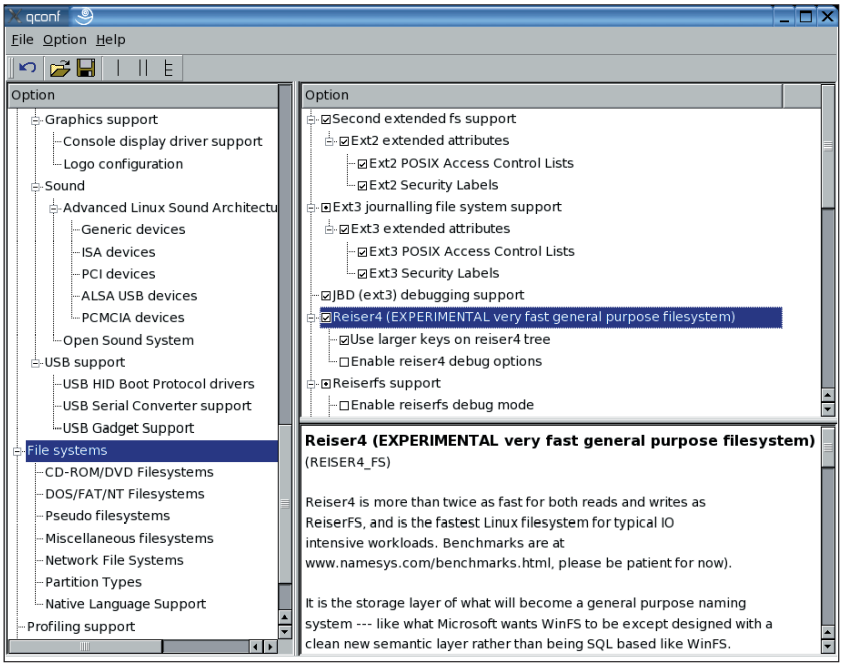

Abbildung 1: Im gepatchten Kernel ist Reiser 4 unter »File systems« zu finden. Das Häkchen bedeutet, dass das System fest in den Kernel integriert wird. Durch einen Doppelklick mutiert das Häkchen zum Punkt. Reiser 4 lässt sich dann als Kernelmodul nachladen.

Installationsanleitung

Diese Anleitung bezieht sich auf Kernel 2.6.5 und die offiziellen Reiser-4-Snapshots vom 26. März 2004 [2]. Bei Patches zu neueren Kernelversionen [3] ist Vorsicht geboten. Sie arbeiten nicht immer fehlerfrei.

- Laden Sie die Kernelquellen vom offiziellen Server [4] und entpacken sie im Verzeichnis »/usr/src/«.
- Kopieren Sie das Patch »all.diff.gz« von der Namesys-Homepage in das Verzeichnis des neuen Kernels »/usr/src/linux-2.6.5/«.
- Wechseln Sie ins Kernelverzeichnis und wenden das Patch mit dem Befehl »zcat all.diff.gz | patch -p1« an.
- In der Datei »fs/reiser4/as_ops.c« fehlt in Zeile 570 der Code »->host«. Die korrekte Zeile lautet:

```
mapping->host->dirtied_when = ↵
jiffies|1;
```

- Ersetzen Sie in der Datei »fs/reiser4/plugin/tail.c« die Zeile 477 »mapping->dirtied _when = jiffies|1;« durch den korrigierten Eintrag:

```
minode->dirtied_when = jiffies|1;
```

- Starten Sie als Root ein Kernel-Konfigurationsprogramm, etwa »make xconfig«. (QT-3-Entwicklerpakete sind erforderlich.)
- Unter »File systems« sollte »Reiser4« bereits ausgewählt sein. Es funktioniert wahlweise statisch oder als dynamisch ladbares Modul, siehe Abbildung 1.
- Nach dem Speichern der Konfiguration kompilieren Sie wie gewohnt den Kernel mit »make bzImage modules modules_install« und kopieren dann den fertigen Kernel in das Zielverzeichnis, zum Beispiel »cp arch/i386/boot/bzImage /boot/vmlinuz-reiser4«. Überschreiben Sie nach Möglichkeit keinen alten Kernel.
- Benutzen Sie zum Booten die Initial RAM-Disk (wie bei Suse), müssen Sie diese mit dem Befehl »mkinitrd« neu anlegen.
- Um den Kernel einfacher zu booten, ist ein Eintrag in der Grub-Konfigurationsdatei »/boot/grub/menu.lst« zu empfehlen.
- Die Datei »/etc/fstab« sollte keine distributionsspezifischen Optionen enthalten, mit denen der Standardkernel nicht zurechtkommt. Unter Suse 9.1 bereiten zum Beispiel die Optionen »acl« und »user _xattr« Probleme. Im Zweifelsfall genügt der Eintrag »rw,exec«.
- Nun können Sie den neuen Kernel booten. Danach müssen Sie noch »libaal-0.5.0.tar.gz« und »reiser4progs-0.5.3.tar.gz« installieren. Das geht wie gewohnt mit »./configure && make && make install«. Sollte Reiser4progs die installierten Libaal-Bibliotheken nicht finden, hilft der Aufruf von »ldconfig«.
- Reiser-4-Partitionen können Sie anschließend mit »/usr/local/sbin/mkfs.reiser4 *Partition*« anlegen und dann mit »mount« einhängen.

Hashwert und einem Erstellungszähler bestehen. Statt dieser noch vom ReiserFS übernommenen Notlösung soll in Zukunft das File-Plugin auch das alphabetische Ordnen von Verzeichnissen mit übernehmen.

Sicherheitsmodule

Eine zentrale Rolle spielen die Sicherheits-Plugins. Das liegt unter anderem am Hauptsponsor von Reiser 4: Die Defensed Advanced Research Projects Agency (DARPA) ist die Forschungsorganisation des amerikanischen Verteidigungsministeriums. Eines dieser Plugins kümmert sich um große Dateien, die viele Informationen enthalten. In herkömmlichen Filesystemen besitzt nur die Datei als Ganzes eine Zugriffsbeschränkung. Es ist zwar mit Access-Control-Listen möglich, verschiedenen Benutzern unterschiedliche Zugriffsmöglichkeiten auf ein und dieselbe Datei einzuräumen, ist jedoch der Zugriff erlaubt, gilt er für das ganze File.

Reiser 4 löst dieses Problem, indem es eine Datei in Elemente (Items) zerlegt, zum Beispiel die Datei »/etc/passwd« in ihre einzelnen Zeilen aufteilt. Jedes dieser Elemente kann eigene Berechtigungen haben und auf verschiedene Plugins zugreifen. Reiser 4 bietet auch die Möglichkeit, je nach Anwendungsbereich ein anderes Trennzeichen (Delimiter) für Item-Grenzen zu verwenden. Anwendungen sehen die einzelnen Elemente dennoch als eine Datei.

Datei mit Unterverzeichnis

Reiser 4 ist – wie oben erwähnt – keine Erweiterung zu ReiserFS, sondern vollständig neu programmiert. Der Theoretiker Hans Reiser legte bei der Entwicklung des neuen Dateisystems großen Wert darauf, den Code so einfach wie möglich zu halten. Zur Entstehung siehe Kasten „ReiserFS, ein Kind der Wende“.

Reiser verzichtete bewusst darauf, das Dateisystem mit vielfältigen Attributen auszustatten, zum Beispiel den erweiterten Attributen unter Ext 2. Jede Funktionalität soll möglichst einfach realisiert werden: über Dateien. Zu diesem Zweck hat unter Reiser 4 jedes Objekt eine Art

eigenen Sub-Namensraum im Verzeichnis »metas«, das der Systemaufruf »readdir« zwar nicht listet, in das man jedoch mit »cd metas« wechseln kann. Das klappt auch für Dateien: »cd *Dateiname*/metas« wechselt in das Metas-Verzeichnis einer Datei. Sie muss dazu jedoch ausführbar sein.

Die Pseudodateien (Abbildung 2) im Verzeichnis »metas« sind Objekte, die an anderen Objekten (Dateien oder Verzeichnisse) hängen. Auf dem Datenträger gibt es sie nicht wirklich (vergleichbar mit den Dateien unter »/proc« oder »/sys«). Mit ihrer Hilfe implementiert Reiser 4 zahlreiche Standard-Unix-Funktionen sowie spezielle Sicherheits- oder Kompressionsverfahren.

So ist es zum Beispiel möglich, ohne »chown«-Aufruf den Eigentümer einer Datei zu ändern oder die Zugriffsrechte eines Verzeichnisses mit einem einfachen »cat«-Befehl abzufragen (Abbildung 2 unten). Eine kurze Beschreibung der wichtigsten Pseudodateien unter »metas« ist in Tabelle 1 zu sehen.

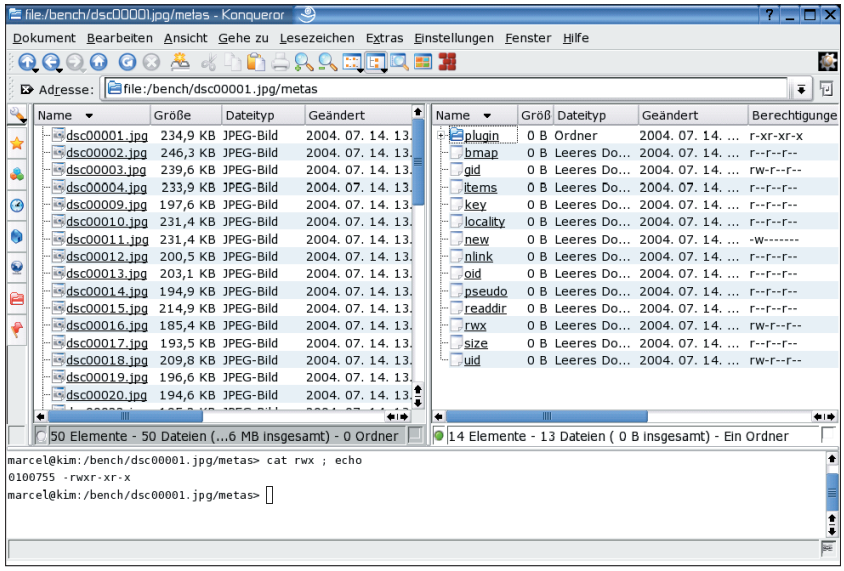

Abbildung 2: Ein Verzeichnis mit Bildern auf einer Reiser-4-Partition (linke Fensterhälfte). Durch Klicken sind die Pseudodateien nicht zu erreichen. Ergänzt man den Pfad in der URL-Zeile um »metas«, so erscheinen die zur Bilddatei »dsc00001.jpg« gehörenden Pseudodateien und das Verzeichnis »plugin« (rechts).

Lebendiges Dateisystem

Moderne Unix-Dateisysteme wie ReiserFS, XFS und JFS basieren auf Baumstrukturen. Dabei ist zwischen den Baumvarianten B+ und B- zu unterscheiden. Während B+ in den internen Knoten nur Zeiger und andere administrative Informationen speichert, legt B- in den internen Knoten auch Daten ab. Da B+ Daten ausschließlich in den Blättern des Baums speichert, muss ein B+-Baum mehr Blätter haben, um die gleiche Anzahl Daten aufzunehmen wie B-. Als Ausgleich lassen sich die internen Knoten effektiver verwalten, da sie nur Zeiger enthalten.

ReiserFS und Reiser 4 verwenden B+-Bäume. Der Unterschied zwischen den traditionellen ausbalancierten Bäumen, wie sie ReiserFS 3.6 verwendet, und den so genannten tanzenden Reiser-4-Bäumen (Dancing Trees) besteht in der Behandlung von BLOBs (Binary Large Objects) und internen Knoten.

BLOBs sind Dateien, die mehr Platz beanspruchen als in einen Endknoten (Blatt) passt. Beim Speichern von BLOBs verwandeln traditionelle Datenbanken und ReiserFS ein Blatt in einen Knoten, der Zeiger auf die neuen Blätter enthält. Dabei entstehen zusätzliche interne Knoten und der High Balanced Tree (der Weg zu allen Blättern von der Wurzel ist gleich lang) gerät aus der Balance, siehe Abbildung 3 oben.

Reiser-4-Bäume speichern BLOBs direkt in den Blättern, sie behandeln BLOBs wie die übrigen Daten (siehe Abbildung 3 unten). Dadurch bleibt die Anzahl

Tabelle 1: Wichtige Pseudodateien

Datei/Verzeichnis	Beschreibung	Erläuterung
bmap	Liste der Blocknummern, die dem Objekt zugeordnet sind	Eine Blocknummer pro Zeile
gid	ID der primären Gruppe des Objekts	Beispiel: 500
items	Liste der Elemente, aus denen das Objekt besteht	Beispiel: »(2a:4:666f6f31000000:0:6d352:2) body length: 3«
key	Schlüssel des Objekts	Beispiel: »2a:4:666f6f31000000:0:6d352:0«
locality	Major Locality des Objekts	Für alle Objekte innerhalb eines Verzeichnisses gleich
new	Pseudodatei zum Anlegen neuer Objekte	
nlink	Anzahl Hardlinks des Objekts	
oid	Objekt-ID des Objekts	Beispiel: »97549«
pagecache	Cache-Statistik des Objekts	Die Datei kann als Ganzes gelesen werden oder als Verzeichnis mit den Unterobjekten »clean«, »dirty«, »io«, »locked« und »nrpages«
plugin	Verfügbare Plugins zum Objekt	Die Liste lässt sich auch per »mkfs.Reiser4 -l« ausgeben
pseudo	Liste der Pseudodateien des Objekts	
readdir	Liste der dem Objekt zugeordneten Unterobjekte	Listet bei Verzeichnissen den Inhalt des Verzeichnisses
rwx	Zugriffsrechte des Objekts in numerischer Form und als Klartext	Beispiel: »0100755 -rwxr-xr-x«
size	Größe des Objekts in Byte	Ein neues leeres Verzeichnis belegt unter Reiser 4 nur 2 Byte plus 1 Byte für jedes neue Unterobjekt
uid	ID des Besitzers des Objekts	Beispiel: »500«

der internen Knoten klein und der Baum bleibt ausbalanciert. Reiser erwähnt auf seiner Homepage, dass die Quellen des Kernels 2.4.18 unter ReiserFS 1629 interne Knoten beanspruchen, unter Reiser 4 hingegen nur 164.

Auf einem durchschnittlichen Computer sollen bei Reiser 4 alle internen Knoten des Baums im Hauptspeicher Platz finden. Ist das Auslagern auf die Platte nicht mehr zu vermeiden, schiebt Reiser 4 zunächst alle Knoten möglichst weit auf die linke Baumseite, um dann im frei gewordenen Platz weitere Knoten von rechts zu holen.

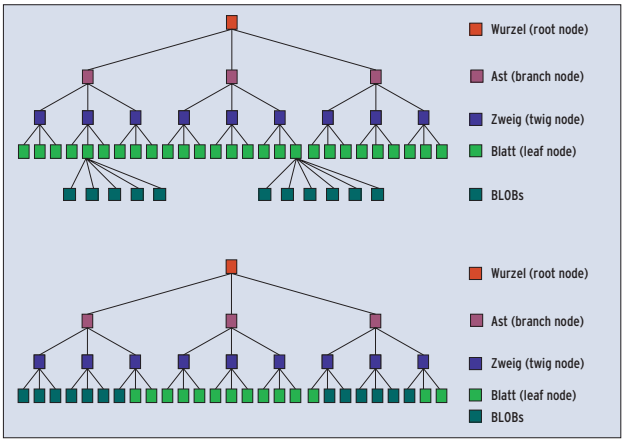

Abbildung 3: Ein traditionell balancierter Baum (oben), wie ihn ReiserFS 3.6 benutzt, speichert BLOBs gesondert. Dazu muss ein Blatt die Funktion eines internen Knotens erfüllen. Ein Reiser-4-Baum (unten) legt BLOBs dagegen in die Blätter (Extents). Damit bleibt die Anzahl der internen Knoten kleiner.

Baum in Tanzfieber

Den Baum fortlaufend auszubalancieren wäre zu aufwändig, solange die Daten im Hauptspeicher liegen. Aber auf der Festplatte sollte er möglichst wenig Platz belegen, um I/O-Aktionen zu sparen. Daher komprimiert Reiser 4 die Daten aus dem Hauptspeicher erst unmittelbar bevor es sie auf die Platte schreibt. Diese Strategie bringt Performance. Traditionell ausbalancierte Bäume, wie sie auch ReiserFS verwendet, sind in der Regel kompakter. Dazu ist aber mehr Verschiebearbeit nötig. Die geringere Kompaktheit ist ein Nachteil der tanzenden Reiser-4-Bäume. Abhilfe soll hier ein Repacker schaffen, der allerdings erst für Version 4.1 geplant ist.

Während ReiserFS neuen Knoten schon beim Anlegen eine Blocknummer zuordnet, wartet Reiser 4 damit so lange, bis das System Daten aus dem Hauptspeicher auf Platte schreibt. Durch dieses von XFS übernommene Feature hat eine neue Datei, die vor dem ersten Speichern gelöscht wird, keine Auswirkung auf das Dateisystem. Auch das wirkt sich positiv auf die Geschwindigkeit aus.

Atomare Transaktionen

Ein alter Wunsch an Dateisysteme sind atomare Transaktionen. Verschiebt der Benutzer eine Datei von A nach B, dann ist die Datei nach einem Stromausfall entweder noch ganz in A oder bereits ganz in B, aber auf keinen Fall sind Teile davon in A und andere Teile in B. Bei

ReiserFS, ein Kind der Wende

Über acht Jahre arbeitete Hans Reiser an seiner Theorie von Datenbanken und Namensräumen, bis er schließlich über die Theorie von „Roads and Waterways“ zu der Erkenntnis kam, dass das Dateisystem für den Computer die gleiche Bedeutung hat wie Straßen und Wasserwege für die Zivilisation. Je besser ein System verknüpft ist, je mehr Möglichkeiten der Interaktion es hat, desto vorteilhafter kann es sich entwickeln.

Reiser ist nicht in erster Linie ein Programmierer, der ein Dateisystem schaffen will, sondern ein Theoretiker, der die beste Anwendung seiner Theorie bei einem Dateisystem sieht. Da Reiser sein Filesystem von Beginn an auf und für Linux entwickeln wollte und ihm nur beschränkte Mittel zur Verfügung standen, sah er sich nach bezahlbaren und qualifizierten Programmierern um. So begann die eigentliche Programmierarbeit an ReiserFS im Jahre 1993 mit Hans Reiser und einem Team junger russischer Entwickler.

Seit Kernel 2.4.21 gilt ReiserFS 3.6 als stabil. Mit Reiser 4 hat die Entwicklerfirma Namesys bereits 2003 intensive Crashtests durchgeführt. Erst jetzt, nachdem die dabei entdeckten Fehler beseitigt sind, steht es für Alltagstests zur Verfügung.

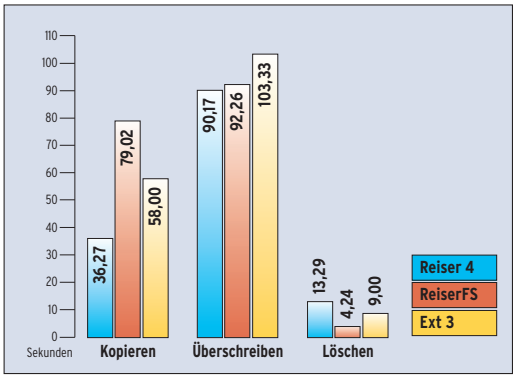

Abbildung 4: Beim erstmaligen Schreiben der Kernelquellen ist Reiser 4 fast doppelt so schnell wie Ext 3. Beim Überschreiben ist das Testfeld ausgeglichener. ReiserFS löscht am schnellsten.

den derzeitigen Lösungen prüft nach einem Ausfall das »fsck«-Programm des jeweiligen Dateisystems die Festplatte und speichert die gefundenen Dateistücke im Verzeichnis »lost + found«.

Reiser 4 verwirklicht alle internen Dateibewegungen atomar. Nach einem Absturz können zwar unvollständige Dateien vorhanden sein (Systemausfall während eines Kopiervorgangs), diese enthalten jedoch ausschließlich Daten aus der kopierten Datei und keinen Datenmüll. Das ist sogar sicherheitsrelevant, da sich im Datenmüll private Informationen befinden könnten.

Reiser 4 schreibt Daten je nach Bedarf einmal (von der Quelle zum Ziel) oder zweimal (von der Quelle in das Journal und von dort zum Ziel). Um auch bei einmaligem Schreiben atomare Transaktionen zu verwirklichen, vertauscht es trickreich das Journal mit dem Ziel.

Soll es eine Datei vom Verzeichnis A nach B kopieren, schreibt Reiser 4 sie zuerst nach LOG, also den Ort, an dem sich das Journal befindet. Dann ändert es die Knoten, die bisher auf B verwiesen, sodass sie nun auf LOG zeigen. Das so neu angelegte Journal wandert an einen neuen Platz – daher der Name Wandering Log.

Beim Reiser-4-System bleiben während einer Transaktion alle geänderten Blöcke an ihrem Ort, bis das System den Abschluss der Transaktion meldet. Ist diese Meldung eingegangen, führt Reiser 4 die Transaktion aus. Ob das Dateisystem dabei ein- oder zweimal schreibt, hängt von den Änderungen ab.

Bei kleinen Änderungen in größeren Dateien ist es sinnvoll, zweimal zu schreiben, da dann die Datei an ihrem Ort bleibt und nur die Änderungen zurückgeschrieben werden müssen. Bei größeren Änderungen ist es schneller, die Knoten zu ändern und die übrige Arbeit dem Repacker zu überlassen. Dieses Feature ist allerdings mangels Repacker noch nicht gegeben. Zur Zeit schreibt Reiser 4 immer zweimal.

Performance

Reiser 4 musste sich als Allzweck-Dateisystem mehreren Benchmarks auf einem gewöhnlichen Desktop-PC stellen. In dem Testrechner steckten 256 MByte Hauptspeicher und eine 40 GByte große Festplatte (Maxtor Baracuda, ATA133). Als Testsystem kam Suse Linux 9.1 mit Kernel 2.6.7-mm4 zum Einsatz, außerdem ein aktuelles Reiser-4-Patch vom 6. Juli 2004 [3].

Als Filesystem-Benchmark diente der »fs_bench« von [5], der die Benchmarks Bonnie++ [6] und Iozone [7] kombiniert. Genauere Informationen zu den Benchmarks gibt es auch unter [9]. Diesen Test ergänzen die Ergebnisse des Benchmark »slow.c« von der Namesys-Homepage sowie eigene Tests.

Reiser wird seinem Ruf gerecht und glänzt vor allem bei kleinen Dateien mit den weitaus besten Benchmark-Ergebnissen (siehe Abbildung 4). Das neue Dateisystem ist nicht nur schnell, sondern spart außerdem Platz. So belegt der kompilierte Quellcode des Kernels 2.6.7 unter Reiser 4 nur 371 MByte. Bei ReiserFS sind es schon 431 MByte und Ext 3 braucht sogar 446 MByte Platz auf der Platte, um die gleiche Menge an Daten abzulegen.

Auffallend ist, dass die Festplatte während der Tests mit Reiser 4 wesentlich leiser arbeitete als unter ReiserFS oder Ext 3. Es ist den Namesys-Entwicklern gelungen, ihre Tests zur Optimierung der Festplatten-Kopfbewegung in die Praxis umzusetzen und Reiser 4 für Schreibzugriffe zu optimieren [8].

Reiser ist Kopiermeister

Wie Abbildung 4 zeigt, schlägt Reiser 4 die Konkurrenz beim Kopieren der Kernelquellen deutlich. Beim Überschreiben sind die Ergebnisse aller drei getesteten Dateisysteme weitgehend ausgeglichen, doch auch hier ist Reiser 4 etwas schneller als die Konkurrenz.

Beim Löschen ist hingegen ReiserFS der schnellste Teilnehmer, Reiser 4 hat bei dieser Aufgabe stark an Boden verloren. Das hängt mit dem internen Aufbau des Dateisystems zusammen und kommt ihm beim Löschen von Dateien in GByte-Größe zugute: Reiser 4 braucht zum Löschen einer 6 GByte großen Datei

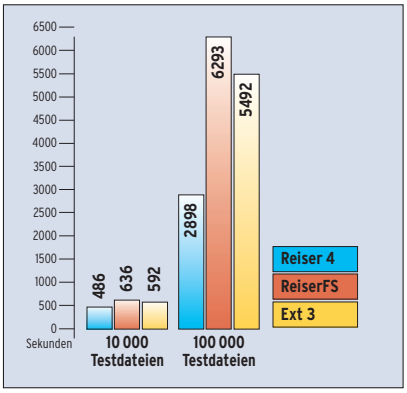

Abbildung 5a: Je mehr Files die 1 GByte Daten enthalten, desto schneller absolviert Reiser 4 den Bonnie++ im Vergleich zu anderen Dateisystemen.

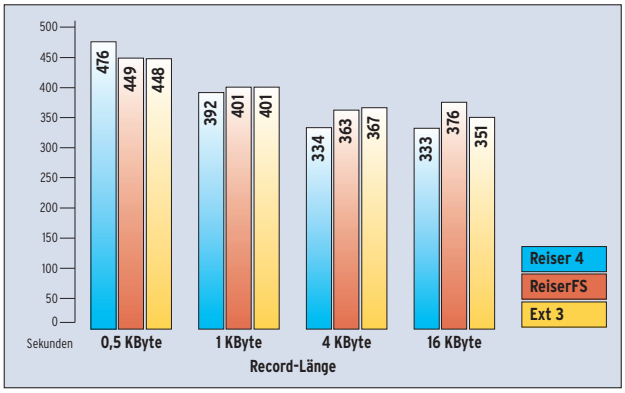

Abbildung 5b: Iozone testete mit einer 1-GByte-Datei. Bei den kleinsten Records ist Reiser 4 zwar noch langsamer als ReiserFS oder Ext 3, in der Praxis sind aber Größen von 4 KByte oder 16 KByte relevant.

gerade mal 1,5 Sekunden. Bei ReiserFS sind es gut 12 Sekunden, Ext 3 brauchte im Test sogar 14 Sekunden.

Bei den Benchmarks Bonnie++ und Iozone sieht das Bild gemischt aus. Reiser 4 vermag zwar in vielen Bereichen zu glänzen, es gibt aber noch Stellen, in denen es weit hinter ReiserFS zurückliegt. Ein Beispiel ist das Löschen in den Rubriken Random Create und Sequential Create unter Bonnie++. Kein gutes Bild bietet Reiser 4 weiterhin bei der Prozessorauslastung, die meisten Testwerte liegen hier klar über Ext 3, teilweise sogar über ReiserFS.

Ein guter Richtwert für die Ausgeglichenheit eines Dateisystems ist die Zeit, die alle Tests zusammen benötigen. Diese Ergebnisse sind in den Abbildungen 5a und 5b zu sehen. Wird nur die Gesamtzeit betrachtet, geht Reiser 4 aus beiden Benchmarks als Sieger hervor.

Schön schnell

Der Benchmark »slow.c« zeigt in erster Linie, dass Reiser 4 gegenüber ReiserFS stark an Performance gewinnt, wenn es gleichzeitig mehrere Datenströme schreiben muss (Abbildung 6). Im Vergleich zum Schreiben oder Lesen einer einzelnen Datei verliert ReiserFS beim parallelen Lesen oder Schreiben deutlich an Leistung. Die Performance von Reiser 4 bleibt dagegen praktisch gleich.

Ext 3 kann beim Schreiben noch gut mit Reiser 4 mithalten, aber beim gleichzeitigen Lesen von vier Datenströmen verliert es deutlich Boden. Reiser 4 kommt sogar der maximalen Durchsatzrate der Festplatte von 27,6 MByte (laut »hdparm -t«) erstaunlich nahe. In Zukunft soll das Dateisystem mit einem Komprimierungs-Plugin auch hier Rekorde aufstellen und die physikalische Durchsatzrate der Festplatte übertreffen.

Nur die wenigsten Benchmarks prüfen Mount-Zeiten und die zum Anlegen des Dateisystems benötigte Zeit. In der Praxis kann genau dies aber für die Wahl des Filesystems ausschlaggebend sein. Bei den Mount-Zeiten hat Reiser 4 noch Nachholbedarf, es braucht rund 12 Sekunden für eine 200-GByte-Partition. Bei ReiserFS sind es 4 Sekunden, Ext 3 braucht dagegen nicht einmal 1 Sekunde, um die gleiche Partition zu mounten. Dafür legt Reiser 4 die Partition in knapp 1 Sekunde an. ReiserFS braucht rund 4 Sekunden.

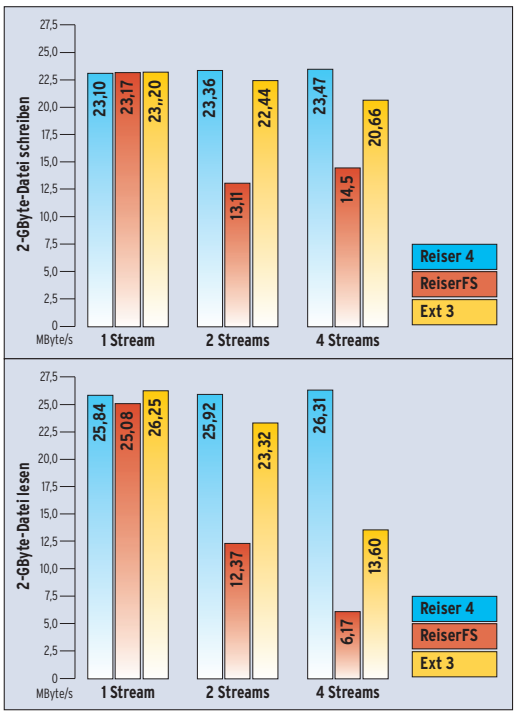

Abbildung 6: Der »slow.c«-Benchmark testet die Geschwindigkeit des Dateisystems beim gleichzeitigen Schreiben mehrerer Datenströme. Reiser 4 büßt kaum Performance ein, Ext 3 nimmt leicht an Leistung ab und ReiserFS wird sogar deutlich langsamer.

Der Unterschied ist damit zu erklären, dass Reiser 4 lediglich die Knoten anlegt, die das Dateisystem zu diesem Zeitpunkt wirklich braucht. Es belegt dann auch kaum Platz auf der Platte. Dagegen legt ReiserFS bereits den Bereich für die Logdateien an und belegt in der Grundeinstellung rund 30 MByte auf der neu erstellten Partition. Beim Anlegen einer Ext-3-Partition von 200 GByte schließlich fühlt man sich wie vor einem DOS-Rechner. Um die Ext-2-Inode-Tabelle und die Logdateien anzulegen, benötigt Ext 3 die stolze Zeit von beinahe 5 Minuten.

Zum Abschluss musste das Reiser-4-Dateisystem mehrere Crashtests bestehen. Der Testrechner wurde mehrmals bei unterschiedlichen Aufgaben über den Reset-Knopf hart gebootet. Auch nach 15 Reboots zeigte Reiser 4 keine Schwächen, alle getesteten Dateien waren problemlos lesbar und das Filesystem intakt.

Reiser 4 bringt viele neue Features und die meisten Benchmarks sehen viel versprechend aus. Die Frage ist weniger, ob Reiser 4 die Erwartungen der Linux-Benutzer erfüllen wird, sondern vielmehr wann. Fast zwei Jahre lang war auf der Namesys-Homepage zu lesen, dass Reiser 4 „in diesem Sommer“ veröffentlicht werde. Jetzt – nach Beendigung der internen Crashtests – braucht es vor allem viele externe Tests und Feedback von der Community. Linspire hat schon angekündigt, Reiser 4 so bald wie möglich in seine Distribution zu integrieren, auch Suse möchte Reiser 4 mitliefern, sobald es einen akzeptablen Stabilitätsgrad erreicht hat.

Weihnachtsgeschenk

Derzeit sucht Namesys nach Sponsoren, da es die DARPA-Gelder nur in die Entwicklung sicherheitstechnischer Features investieren darf. Wer also Reiser 4 als Weihnachtsgeschenk auf seinem Computer installieren möchte, sollte vielleicht auch mal über ein Geschenk an Namesys nachdenken. *(fjl)*

Infos

[1] Reiser: [http://www.namesys.com]
[2] Snapshot für Kernel 2.6.5: [http://www.namesys.com/snapshots/LATEST/]
[3] Patches zum aktuellen Kernel: [http://www.namesys.com/auto-snapshots/]
[4] Linux-Kernel: [http://www.kernel.org]
[5] Fsbench: [http://fsbench.netnation.com]
[6] Bonnie++: [http://www.coker.com.au/bonnie++/]
[7] Iozone: [http://www.iozone.org]
[8] Reiser-4-Design: [http://www.namesys.com/v4/reiser4_the_atomic_fs.html]
[9] Mirco Dölle und Jörg Reitter, „Performance-Vergleich von Ext 2/3, JFS, ReiserFS und XFS“: Linux-Magazin 03/04, S. 38

Der Autor

Marcel Hilzinger studierte an der Universität Zürich Germanistik und Osteuropäische Geschichte. Ab 2001 arbeitete er über drei Jahre für das ungarische Büro von Suse Linux in Budapest und übersetzte unter anderem die Suse-Dokumentation ins Ungarische.