SelfLinux-0.13.1

Journaling Dateisysteme

Autor: Florian Frank (*florian@pingos.org*)
Autor: Jörn Bruns (*joern_bruns@gmx.de*)
Formatierung: Matthias Hagedorn (*matthias.hagedorn@selflinux.org*)

Inhaltsverzeichnis

1 Vorbemerkung

Ein Dateisystem ermöglicht das Ablegen und Bearbeiten von Dateien und Verzeichnissen auf einem Datenträger. Damit ist hier nicht die Hierarchie des UNIX-Verzeichnisbaumes mit /var, /usr, etc. gemeint, sondern die unterliegende Struktur, um die Daten physikalisch auf dem Datenträger abzulegen.

Für Linux gibt es inzwischen eine größere Auswahl an Dateisystemen mit jeweils verschiedenen Vor- und Nachteilen. Da wären einerseits das traditionelle Ext2, andererseits die Journaling Dateisysteme Ext3, ReiserFS, JFS und XFS. Es gilt nun, für die jeweilige Nutzung das am besten geeignete auszuwählen.

In diesem Artikel wird die Bezeichnung **Journaling-Dateisystem** verwendet. Deutsche Übersetzungen wie etwa **protokollierendes Dateisystem** haben sich bislang nicht durchsetzen können und würden daher möglicherweise ein zügiges Auffinden des Kapitels verhindern.

2 Wichtige Eigenschaften eines Dateisystems

2.1 Journaling

Dies ist die wohl wichtigste Eigenschaft der neueren Dateisysteme. Ein Journal ermöglicht es, ein Dateisystem nach einem plötzlichen Systemausfall in einem konsistenten Zustand zu erhalten. Damit sind langwierige Dateisystem-Tests nach einem solchen Ausfall nicht mehr notwendig.

2.1.1 Woher kommen inkonsistente Zustände nach einem Systemausfall?

Daten werden nicht sofort auf den Datenträger geschrieben, sondern aus Performance-Gründen zunächst im Arbeitsspeicher gehalten. Für die Anwendungen gelten die Daten aber schon in diesem Zustand als gespeichert, damit diese zügig weiterarbeiten können. Im Arbeitsspeicher wird zusätzlich die Reihenfolge der Schreibzugriffe so umgestellt, dass möglichst viele Schreibzugriffe auf einmal durchgeführt werden können. Durch die Firmware der Festplatten wird die nochmals optimiert, so dasse Kopfbewegungen der Festplatte erheblich reduziert werden.

Die Daten werden anschließend, mit einer gewissen zeitlichen Verzögerung, in einem Rutsch auf die Festplatte geschrieben. Dieses als Caching bezeichnete Verfahren ermöglicht ein wesentlich schnelleres Arbeiten.

Fällt nun aber plötzlich der Strom aus, ist nicht klar, in welchem Zustand die Daten gerade waren. Sind sie auf die Platte geschrieben oder waren sie noch im Arbeitsspeicher?

Deshalb ist in solch einem Fall ohne Journaling eine Prüfung aller Dateien notwendig, was bei größeren Festplatten sehr lange, bis zu mehrere Stunden, dauern kann. Dies ist für Produktiv-Systeme in der Regel nicht akzeptabel.

Darüber hinaus kann bei Inkonsistenzen ein manueller Eingriff notwendig werden, schlimmstenfalls lässt sich das Dateisystem nicht mehr reparieren, was allerdings sehr selten vorkommt.

Das Journaling-Dateisystem vermeidet derartig lange Dateisystem-Prüfungen. Darüber hinaus werden die genannten Inkonsistenzen, die in seltenen Fällen das Dateisystem zerstören können, meist vermieden.

2.1.2 Wie funktioniert ein Journaling-Dateisystem?

2.1.2.1 Metadaten

Ein Dateisystem benötigt interne Verwaltungs-Strukturen, welche die eigentlichen Daten der Festplatte organisieren und griffbereit halten. Solche internen Strukturen werden **Metadaten** genannt und sind sozusagen die Daten über die Daten. Die Metadaten definieren z. B., wo die Datenblöcke einer Datei zu finden sind, wer Besitzer ist, die Rechte, die letzten Zugriffszeitpunkte und anderes mehr.

Diese Verwaltungsdaten müssen unbedingt konsistent gehalten werden. So lässt sich auf eine Datei nicht zugreifen, wenn die Datenblöcke nicht dort sind, wo sie laut Metadaten zu sein haben. Oder es könnte passieren, dass bestimmte Datenblöcke als nicht belegt definiert sind, obwohl dort Daten abgelegt sind, die somit überschrieben werden könnten.

Wird eine Datei neu angelegt, so werden in mindestens fünf verschiedenen Strukturen der Metadaten Änderungen vorgenommen. Gibt es während dieser Änderungen einen Systemausfall, ist das Dateisystem inkonsistent - es sei denn, es gibt ein Journal.

2.1.2.2 Journal

Bevor eine Änderung an den Metadaten vorgenommen wird, wie durch das Anlegen einer neuen Datei, werden die dafür nötigen Metadaten-Änderungen zunächst ausschließlich in das Journal geschrieben, welches eine Art Log-Datei darstellt. Diese Einträge im Journal gelten solange nicht für das Dateisystem, bis die Journal-Einträge mit einem **commit** abgeschlossen werden. Erst dann werden die neuen Metadaten auf die Festplatte geschrieben.

Wie soll dies nun vor Inkonsistenzen nach einem Systemabsturz schützen? Nach einem Neustart zieht das Dateisystem als erstes das Journal zu Rate. Sind die Einträge im Journal schon mit einem **commit** abgeschlossen, sind die Metadaten gültig und die Einträge werden auf die Festplatte übertragen. Fehlt das **commit** als abschließender Eintrag, werden die Metadaten nicht von dem Journal auf die Festplatte geschrieben, sondern verworfen.

Bei Dateisystemen ohne Journal, wie **Ext2**, müssen dagegen alle Metadaten überprüft werden, ob sie konsistent sind, was die erwähnten langen Wartezeiten bewirkt.

Was ist nun mit den eigentlichen Daten, wann werden diese auf die Festplatte gespeichert? Das ist bei den verschiedenen Dateisystemen verschieden implementiert. Bei **Ext3** werden zunächst die eigentlichen Daten auf die Festplatte geschrieben, erst anschließend wird das abschließende **commit** im Journal gesetzt. Bei den anderen Journaling Dateisystemen können dagegen die Metadaten schon auf die Festplatte geschrieben werden, bevor die Daten komplett auf der Festplatte sind, was zu Problemen führen kann, aber schneller ist. Hier hat **Ext3** in Sachen Sicherheit die Nase vorn.

Die Integrität der eigentlichen Daten stellt das Journal leider nicht sicher. Es kann also durchaus sein, das nach einem Absturz eine Datei einen Mix aus einer alten und neuen Version enthält. Auch hier hat **Ext3** die Nase vorn, denn nur **Ext3** kann über die Metadaten hinaus auch die Datenänderungen selbst mitprotokollieren, mehr dazu siehe Ext2 und Ext3.

2.2 Binärbäume

Ein Binärbaum ermöglicht einen beschleunigten Zugriff auf die Informationen der Verzeichniseinträge, insbesondere bei Verzeichnissen mit vielen Dateien und Unterverzeichnissen. Bei einem traditionellen Dateisystem wie **Ext2/Ext3** müssen der Reihe nach alle Verzeichniseinträge durchgegangen werden, bis der gesuchte Eintrag gefunden ist (doppelt verzeigerte Liste). Somit ermöglicht ein Binärbaum ein beschleunigtes Suchen im Dateisystem.

Beispiel:

Wird ein Verzeichnis mit 1000 Einträgen nach einem Dateinamen durchsucht, sind ohne Binärbaum-Struktur durchschnittlich 500 Suchaktionen notwendig, mit Hilfe eines Binärbaum dagegen nur 10.

Allerdings benötigt ein Binärbaum mehr Rechenzeit, da das System komplexer ist und nach Bearbeitung der Verzeichniseinträge erneut ausbalanciert werden muss.

2.3 Flexibleres setzen der Rechte durch ACLs (Access Control Lists)

Außer **ReiserFS** können alle hier genannten Dateisysteme um ACLs erweitert werden.

Der Vorteil liegt im Wegfall der Unix-typischen Beschränkung, das Rechte auf eine Datei oder ein Verzeichnis nur für je einen Besitzer, eine Gruppe und alle anderen gesetzt werden kann. Im Gegensatz dazu können mit Hilfe von ACLs verschiedene Rechte für nahezu beliebig viele Nutzer und Gruppen vergeben werden, was insbesondere für Netzwerkanwendungen sehr nützlich ist. So kann einer Gruppe Lese-, einer anderen Gruppe

Schreibrecht gegeben werden, alle anderen können ausgeschlossen werden.

Es können allerdings bislang nur wenige Linux-Programme diese erweiterten Rechte nutzen, wie etwa Samba.

2.4 Geschwindigkeit

Es sind nicht wenige Geschwindigkeitstests veröffentlicht worden. Je nach Testumgebung und Zeitpunkt unterscheiden sich die Ergebnisse erheblich, widersprechen sich zum Teil, deshalb müssen sie mit großer Vorsicht genossen werden.

Außerdem entwickeln sich die neueren Dateisysteme rasant, so dass die Tests oft schon veraltet sind. Und es ist die Frage, wie relevant die Geschwindigkeit für die Praxis ist. Deshalb sind hier nur allgemeine Tendenzen der Dateisysteme angegeben.

2.5 Regelmäßige Dateisystem-Check

Journaling verhindert leider nicht die regelmäßigen Dateisystem-Überprüfungen, wie oft behauptet wird. Ist der **maximum mount count** (oft nach dem 24. mounten) oder aber der **check intervall** (oft nach einem halben Jahr ohne Dateisystem-Check) überschritten, löst ein Neustart eine Dateisystemprüfung aus, was natürlich oft sehr störend ist. Es sollen damit einerseits Fehler im Dateisystem-Code begegnet, andererseits frühzeitig Hardware-Fehler erkannt werden, wie etwa defekte Sektoren.

3 Verschiedene Dateisysteme für verschiedene Zwecke

Alle beschriebenen Dateisysteme sind im Kernel 2.6 und ab Kernel 2.4.25 integriert.

3.1 Ext2 und Ext3

Bis vor kurzem war **Ext2** das einzig ernsthaft nutzbare Linux-Dateisystem. Es ist sehr stabil, da es seit 1993 für Linux entwickelt und korrigiert wird. **Ext2** enthält keine Journal-Funktion und muss ohne Binärbaum auskommen, **Ext3** ist ein **Ext2**, das um das Journaling erweitert wurde. **Ext2** kann in **Ext3** umgewandelt werden und umgekehrt. Diese Konvertierung ist zwischen anderen Dateisysteme nicht möglich.

Die Geschwindigkeit scheint akzeptabel zu sein für die klassische, **unmoderne** Architektur des Dateisystems. Nur bei sehr große Dateien im GB-Bereich sowie bei Verzeichnissen mit tausenden von Dateien scheint Ext2/3 von der Konkurrenz deutlich abgehängt zu werden.

Wie bereits weiter oben erwähnt, hat **Ext3** zwei Funktionen, welche die anderen **modernen** Dateisysteme nicht haben.

Einmal ist das Zusammenspiel von Metadaten und Journal exakter synchronisiert, denn das **commit** wird erst in das Journal geschrieben, wenn die eigentlichen Daten wirklich auf der Festplatte sind. (Dies ist zur Beschleunigung übrigens deaktivierbar mit der Mount-Option `data=writeback`.)

Zum anderen kann **Ext3** als einziger Kandidat auch die Daten selbst in sein Journal eintragen, was aber meistens einen starken Geschwindigkeitseinbruch bewirkt. Um diese Funktion zu aktivieren, muss die Mount-Optionenn `data=journal` in die /etc/fstab eingetragen werden. Wird dies so eingerichtet, empfiehlt sich zur Beschleunigung das Auslagern des Journals auf eine weitere Festplatte.

Des Weiteren hat **Ext2/3** die meisten ergänzenden Zusatz-Programme.

Wenn es also darum geht, ein sehr zuverlässiges Linux aufzusetzen und die Geschwindigkeit des Dateisystems nicht das Allerwichtigste ist, scheint **Ext3** eine gute Wahl zu sein.

3.2 ReiserFS

ReiserFS wurde als einziges der hier erwähnten Journaling Dateisysteme komplett neu für Linux geschrieben. Es ist das neueste Dateisystem, gleichzeitig ist es jedoch auch das am längsten für Linux verfügbare Journaling-Dateisystem. Auch wenn es in den Anfangszeiten kein Journaling bereitstellte.

Es nutzt das Binärbaum-Konzept nicht nur für die Verzeichniseinträge, sondern teilweise auch für die Daten selbst, was kein anderes der hier aufgeführten Dateisysteme umsetzt.

Durch den konsequenten Einsatz neuer Techniken sollte man annehmen, ReiserFS sei das schnellste aller Dateisysteme. Den Tests nach zu urteilen ist dies aber nicht unbedingt so. Es zeigt sich jedoch, dass **ReiserFS** viele kleine Dateien schneller löschen kann als jedes andere Linux-Dateisystem.

ReiserFS unterstützt im Kernel 2.6 als einziges Dateisystem keine ACLs, was dieses Dateisystem für den Fileserver Samba weniger attraktiv macht, zumindest wenn dort diese erweiterten Rechte genutzt werden sollen.

Es gibt auch nur sehr wenige Hilfs-Programme für **ReiserFS**.

ReiserFS gilt in den neueren Tests als akzeptabel stabil. Es gab in der Vergangenheit mehrere Probleme mit

einigen Anwendungen, wie mit NFS, die aber anscheinend inzwischen behoben sind.

Mit einigen Einschränkungen muss **ReiserFS** leben:

* Die Blockgröße ist auf 4K festgelegt.
* Es ist auf x86-Plattformen beschränkt.
* Quota, also Beschränkung auf eine maximalen Größe, die z. B. ein Home-Verzeichnis haben darf, wird nicht unterstützt.
* Es unterstützt keine ACLs.

3.3 IBM-JFS

Unter OS2 und AIX schon länger bewährt, hat **IBM JFS** auf Linux portiert und unter GPL freigegeben. **JFS** ist Bestandteil des Standard-Kernels ab der Version 2.4.20.

Die anfänglichen Stabilitätsprobleme scheinen überwunden zu sein. Die Entwicklung ist aber noch nicht sehr weit vorangeschritten, so gibt es keine Quotas und die Hilfswerkzeuge, welche unter AIX zur Verfügung stehen, sind unter Linux nur zu einem geringem Umfang verfügbar. Auch die Dokumentation ist bisher eher knapp gehalten.

Unter Performance-Aspekten spielt **JFS** an vorderster Reihe mit.

3.4 SGI-XFS

Auch **XFS** läuft inzwischen sehr stabil. In einigen Tests holt es sich Performance-Siege ein, v.a. beim Kopieren vieler kleiner Dateien als auch im Umgang mit sehr großen Dateien im GB Bereich.

Der Bootmanager `LILO` muss so konfiguriert werden, das er in den MBR (Master Boot Record), nicht in die Root-Partition installiert wird, damit das System bootfähig ist. `GRUB`, ein anderer Bootmanager, kann seit der Version 0.91 mit **XFS** umgehen.

Von den hier vorgestellten Dateisystemen ist **XFS** als letztes in den Kernel 2.4 integriert worden, seit der Version 2.4.25. Im Kernel 2.6 ist es seit dem ersten Release dabei.

XFS hat viele Werkzeuge zur Verwaltung des Dateisystems zu bieten, wie etwa zur Daten-Sicherung inklusive der ACL-Rechte (`xfsdump`), Partitionen vergrößern (`xfs_growfs`) oder Rettungs-Tools. Von der Anzahl und Vielseitigkeit dieser Werkzeuge her kann dem nur **Ext2/3** das Wasser reichen. Außerdem ist die Dokumentation umfangreich.

4 Fazit

Für eine normale Workstation oder einen Laptop ist es nicht sonderlich relevant, welches der genannten Dateisysteme genutzt wird, solange es über Journaling-Funktionalitäten verfügt. Der Anwender wird die Unterschiede selten zu spüren bekommen. Anders sieht es hier im Server-Bereich aus. Hier kann die Wahl des falschen Dateisystems sehr neagtive Auswirkungen auf die Gesamtgeschwindigkeit des Systems haben. Hier sollte man vor allem bei grösseren Installationen unbedingt eigene Tests unter den zu erwartenden Bedingungen durchführen.

Soll Samba mit ACLs genutzt werden, sollte ein Dateisystem um diese Funktionalität ergänzt werden. Nur **ReiserFS** ist dazu derzeit nicht in der Lage und schließt sich somit für diesen Anwendungsfall aus.

Ext2, vor allem aber **Ext3** scheinen noch lange nicht überlebt zu sein. **Ext3** hat einige Funktionen, die kein anderes der hier genannten Dateisystem bieten kann, siehe Ext2 und Ext3 . Vor allem ist **Ext2/3** das bewährteste Dateisystem, das weitgehend fehlerbereinigt ist. Soll also ein möglichst stabiles System eingerichtet werden, könnte **Ext3** die erste Wahl sein.

ReiserFS scheint inzwischen ebenfalls einen einigermassen stabilen Zustand erreicht zu haben und hat sich seit einigen Jahren bewährt. Dennoch liest man immer wieder von extremen Probleme in Bezug auf **ReiserFS**, wenn es zu einem Ausfall gekommen ist. Diese können bis hin zum totalen Datenverlust gehen.

Aufgrund der noch geringen Verbreitung gibt es nicht so viel Hilfe und Erfahrungen mit **JFS** und **XFS** unter Linux.

Die Beurteilungen von **XFS** und **JFS** fallen in den Tests sehr unterschiedlich aus. Dies liegt vermutlich auch an der kurzen Bewährungszeit und deren noch relativ seltenem Einsatz unter Linux. Da der Linux-Kernel nun alle erwähnten Dateisysteme von Haus aus dabei hat, wird die Nutzung von **JFS** und **XFS** unter Linux sicherlich zunehmen, da sie somit in jeder kommenden Linux-Distribution enthalten sind.