FAU Erlangen-Nürnberg
Seminar: Konzepte von Betriebssystemkomponenten
Referent: Johannnes Jordan
Datum: 19.01.2004

Journaling-Dateisysteme

an den Beispielen ext3 und ReiserFS

1. Theoretische Grundlagen des Journaling

Wie arbeiten Dateisysteme?

Es gibt viele verschiedene Möglichkeiten, Dateien anzuordnen, sowohl für den Inhalt selbst, als auch vor allem für die Verwaltungsinformationen. Im alten Unix Fast File System war in der Nummer einer Inode noch umgerechnet enthalten, wo diese genau auf der Platte zu finden ist. Die breite Querverteilung ist nicht gerade performanceträchtig, da z.B. der Lesekopf einer Festplatte für jede Inode springen muss. Heutzutage werden Inodes eher auf größeren Haufen gelagert oder sind in B-Bäumen verschiedener Arten angelegt. Auch die Aufteilung des Speichers lässt sich auf verschiedenste Arten und Weisen erledigen, z.B. aufgeteilt in Blöcke, über deren Zustand (frei oder belegt) dann Bitfelder an fixen Positionen Auskunft erteilen. Der freie Speicher kann aber auch in sogenannten Extends (bestehend aus dem Zeiger auf den Beginn und die Angabe der Länge) verwaltet werden, die dann z.B. auch wieder nach Größe geordnet in einem B-Baum abgelegt sind.
Noch recht einfach gehalten ist die Organisation in ext2. Hier gibt es Blockgruppen, jede für sich wird Superblock des Dateisystems (globale Informationen) angeführt, gefolgt von einem Block-Bitfeld, einem Inode-Bitfeld, der Inodetabelle und dann den Datenblöcken. Verzeichnisse sind verkettete Listen der Dateieinträge.
Auch interessant ist dabei, nach welchen Regeln man neue Daten anordnet, um zum Beispiel Fragmentierung vorbeugen zu können.

Was bedeutet Journaling?

Protokollierung:

Kernpunkt beim Journaling ist das Verwalten eines Logs für die entsprechende Partition. In diesem Log werden nur Schreiboperationen gespeichert. Das Log dient allerdings nicht dazu, generell Protokoll zu führen, denn im Prinzip ist nur wichtig, was gerade im Moment erledigt werden soll. Sobald ein Schreibvorgang abgeschlossen ist, kann seine Protokollierung verworfen werden. Die Informationen im Log werden nur ausgewertet, wenn solche Vorgänge nicht zu Ende geführt wurden.

Log-Structured-Filesystem:

Bei diesem Modell eines Dateisystems geht man noch einen Schritt weiter als bei sogenannten statischen Dateisystemen mit aufgesetztem Journaling. Anstelle die Partition in Blöcken zu strukturieren, die dann Daten oder Inodes zugeordnet werden, dehnt sich ein Quasi-Protokoll linear immer weiter im Speicher aus. In diesem steht dann natürlich alles, Metadaten wie Inhalte. Für den Lesezugriff wird die Suche durch einen Index erleichtert. Schreibvorgänge sind relativ schnell erledigt - man schreibt ja prinzipiell immer nur an einer Stelle und erspart sich damit Zugriffszeiten.

Wieso ist Journaling nützlich/nötig?

Inkonsistenzen:

Auch wenn man in der Nicht-Windows-Welt von Bluescreens eher verschont bleibt, kann es zu verschiedenen Gelegenheiten zur Korrumpierung der Daten kommen, z.B. bei Stromausfällen oder Hardwareproblemen. Dabei ist das Hauptproblem das verzögerte Abwickeln von Schreiboperationen zur Performancegewinnung. Schreiboperationen werden gerne zusammengefasst, da sie dann „in einem Rutsch" meist schneller vonstatten gehen. Außerdem können sie ausgeführt werden, wenn das System gerade nicht ausgelastet ist. Im Falle eines Ausfalls können nun Inkonsistenzen auftreten:

- **Inode-Referenzen können verloren gehen:** z.B. wird ein Inode angelegt, aber der Datei-Eintrag in einem Verzeichnis dazu fehlt. Oder aber verschobene Dateien werden nun im Quell- und Zielverzeichnis, oder in keinem von beiden, referenziert.
- **Meta-Infos der Inodes können falsch sein:** z.B. an eine Datei angehängte Daten liegen in Blöcken, die bei der Inode noch nicht eingetragen sind. Oder aber, eigentlich belegte Blöcke sind noch als frei gekennzeichnet.
- **Die Daten selbst können zerstört sein:** z.B. ungünstigerweise, wenn gerade Daten überschrieben werden, aber natürlich auch beim Speichern neuer Blöcke.

Die klassische Lösung:

Bei klassischen Dateisystemen ohne Journal müssen solche Fehler sehr aufwändig gesucht und bestmöglichst bereinigt werden. ext2 z.B. besitzt ein sogenanntes „clear"-Flag, welches beim Ein- und Aushängen der Partition entsprechend gesetzt wird. Wurde die Partition nicht korrekt ausgehängt, wird sie automatisch beim Einhängen überprüft, ebenfalls meist routiniert.
Diese Prüfung der Konsistenz kann bei sehr großen Partitionen mit vielen Dateien mehrere Minuten bis Stunden dauern. Dabei müssen schließlich alle Dateireferenzen auf Inodes überprüft werden, ob alle in Inodes (auch indirekt) eingetragenen Blöcke auch richtig als belegt markiert sind und freie Blöcke nicht fälschlicherweise als belegt. Ersteres ist auch umgekehrt nötig - existiert auch für jede Inode die entspr. Anzahl Referenzierungen? Inodes ohne Referenz werden z.B. bei ext2 in „/lost+found" abgespeichert.
Dabei ist dieser Vorgang keineswegs immer erfolgreich - existieren auf der Partition die nötigen Informationen zur Reparatur nicht, so ist diese schlichtweg nicht korrekt möglich.

Die Lösung bei einem Journaling-Dateisystem:

Anstelle die gesamte Konsistenz in Frage zu stellen, reicht es, beim Einhängen der Partition das Protokoll auszuwerten, um eine mögliche Korruption zu erkennen. Je nach Art des Journalings können unterbrochene Operationen wiederhergestellt werden. Auf jeden Fall aber ist die korrekte Organisationsstruktur gewährleistet, in bestimmten Fällen auch die der Daten.

Etwas weitergedacht - atomares Dateisystem:

Unter einem atomaren Dateisystem versteht man eine garantierte Konsistenz auch im Sinne des Transaktionsmodells. Aktionen sollen entweder komplett oder gar nicht durchgeführt werden. Klassisches Beispiel: Wird Geld von einem Konto auf das andere transferiert, so sollte das Geld nur abgebucht werden, wenn es auch auf der anderen Seite eingezahlt wird.
Auch hier ist Journaling notwendig, um auszuschließen, dass nur ein Teil des Vorgangs ausgeführt wird, bzw. um den vorherigen oder gar erwünschten Zustand der Daten zu erreichen.

Welche Möglichkeiten des Journaling gibt es?

- **Logging der Meta-Daten:**
  In den meisten Fällen reicht es aus, im Protokoll nur die Operation zu beschreiben, nicht aber deren Inhalt. So wird z.B. protokolliert, dass eine Inode angelegt wird, dass sie in einem Verzeichnis referenziert wird, welche Rechte sie bekommt, wem sie gehört, etc. Desweiteren natürlich auch, welche Blöcke sie belegen soll. Was in diese Blöcke gespeichert wird, ist im Protokoll nicht enthalten. Sichergestellt wird damit, dass das Dateisystem an sich konsistent bleibt; die gerade behandelten Daten können aber verlorengehen.

  So könnte ein Protokolleintrag aussehen, wenn eine Inode mit der Nummer 777 und den Datenblöcken 3110, 3111 und 3506 verändert und dabei erweitert wird:

```
 Inode 777:  intent-to-commit
Block 3111:  data update (changes)
Block 3506:  data update (changes)
Block 3790:  data update (changes)
Block 3791:  data update (changes)
 Inode 777:  update data block list to
             3110, 3111, 3506,
             3790, 3791
 Inode 777:  access time 23:42
             19-JAN-2003
 Inode 777:  modification time 23:42
             19-JAN-2003
 Inode 777:  committed
```

- **Vollständiges Logging:**
  Unter dem Begriff „Data Journaling“ (oder auch „Full Journaling“) beschreibt man das Protokollieren des gesamten Vorgangs, inklusive der behandelten Daten. Damit erreicht man, dass beim Überschreiben von Daten zumindest das Original nicht verloren geht. Ein atomares Dateisystem ist also unbedingt auf Data Journaling angewiesen.

- **Log-Structured:**
  Diese Dateisystemart hat trotz anderer Methodik den gleichen Effekt wie Data Journaling - alte Daten werden per Definition erstmal nicht überschrieben; zu schreibendes wird schließlich stets angehängt. Bei einem Ausfall muss beim Einhängen nur festgestellt werden, bis zu welchem Punkt alles konsistent ist - dieser Punkt wird als „Checkpoint“ beschrieben. Mit zwei „Checkpoint Regions“ an fixer Position lässt sich der letzte Checkpoint herausfinden. Die Checkpoint Regions werden abwechselnd beschrieben, um auch mit einem Ausfall beim Schreiben des Checkpoints fertigzuwerden. Dabei wird nebst Zeiger auf das letzte beschriebene Segment *(darauf kommen wir noch zurück)* und den globalen Daten (Inode-Map) der Zeitpunkt zuletzt abgelegt. Beim Einhängen wird dann die jeweils aktuellere Region herangezogen. Checkpoints müssen nicht allzuhäufig passieren, denn ab dem Checkpunkt kann das Log bei der Wiederherstellung der Reihe nach ausgewertet werden. Dazu müssen im Log zusätzlich zu erstellten Inode- und Verzeichniseinträgen vorneweg diese angekündigt werden. Ansonsten könnte beim Verlust von einem der beiden durch einen Ausfall der Referenzzähler der Inode nicht mehr stimmen - neue Inode- und Verzeichniseinträge nach einem Checkpoint müssten verworfen werden.
  Dieses Konzept ist wohl als das robusteste zu bezeichnen.

Welche Nachteile hat Journaling?

- **Aufwand der Protokollierung:**
  Nachdem das Log bekanntermaßen nur auf einem Festspeicher Sinn macht, entstehen dadurch merkliche Kosten. Meist liegt es auf dem gleichen Datenträger wie auch die entspr. bearbeiteten Daten und somit muss die Festplatte beim Schreiben zwischen Log und den bearbeiteten Daten hin- und herspringen. Allerdings kann die Führung des Protokolls sehr gut optimiert werden, sodass die entspr. Einträge erheblich schneller erledigt werden können, als die beschriebenen Operationen selbst.

- **Doppeltes Schreiben beim Data-Journaling:**
  Will man atomar arbeiten, ist es konventionellerweise erforderlich, die entsprechenden Daten doppelt zu schreiben. Dabei werden sie erst im Log aufgefüllt, um dann nochmals an ihr wirkliches Ziel geschrieben zu werden. Dabei wird sichergestellt, dass alte Daten nur dann überschrieben werden, wenn die neuen auch wirklich komplett vorhanden sind.

- **Fragmentierung bei Log-Structured:**
  Auch wenn ein Log-Structured-Dateisystem das Journaling ja quasi nebenbei erledigt und somit dadurch auch keinerlei Performance-Einbußen entstehen, gibt es dafür ein anderes großes Problem. Irgendwann ist die Partition voll und man ist auf den Speicherplatz innerhalb des Logs angewiesen, der durch veraltete oder gelöschte Daten belegt ist. Diese „toten" Bereiche einfach wieder abzulaufen, würde wenig Sinn ergeben, also müssen regelmäßig die bisher abgelegten Daten lückenlos neu angelegt werden. Um das Ganze etwas im Rahmen zu halten, ist der Speicherplatz in die oben schon angestoßenen Segmente unterteilt. Bereits zusammengepackte Segmente werden dahingehend als rein gekennzeichnet. Beim Aufräumvorgang werden unreine Segmente zusammengefasst als neue, reine Segmente geschrieben.

2. Implementierungsbeispiele: ext3 und ReiserFS 3 & 4

ext3 - ein erweitertes ext2:

Dieses System wurde als Aufsatz für das unter Linux sehr etablierte und bewährte ext2 konzipiert. Die grundlegende Struktur des Dateisystems bleibt die gleiche und wird nur durch das Journal erweitert. Die Protokollierung kann auf drei verschiedene Arten arbeiten:

- **data=writeback:** Das ist die lässigste Art des Journalings, mit dem resultierenden geringstem Performanceverlust. Es werden lediglich die Metadaten protokolliert, unabhängig vom Schreiben der Daten an sich. Somit bleibt die Verwaltungsstruktur konsistent, aber gerade veränderte Daten können beschädigt werden.

- **data=ordered:** Dies ist die voreingestellte und meist angewendete Arbeitsweise. Auch hier werden nur Metadaten protokolliert, allerdings werden Log, Metadaten und Inhalte synchron geschrieben (zuerst die Inhalte). Das kommt beim Hinzufügen von Daten prinzipiell einem „Full Journaling" gleich. Korrupte Daten entstehen allerdings noch beim Überschreibvorgang. Dabei ist nicht mal sicherstellbar, welche Blöcke überschrieben wurden und welche nicht, da die Reihenfolge der Festplatte (Zwischenspeicher...) überlassen wird.
  Da dieser Fall verhältnismäßig selten auftritt, ist dieser Modus ein sehr guter Kompromiss zwischen dem kostenintensiven Data Journaling und dem blanken Loggen der Metadaten.

- **data=journal:** Hier wird ganz simpel alles protokolliert. Zuerst landen die zu schreibenden Daten im Journal, dann erst werden sie erneut an ihren Zielort geschrieben. Gibt es bei diesem Vorgang einen Ausfall, kann er anhand des Journals rekonstruiert werden.
  Selbstverständlich sind die Kosten durch das doppelte Schreiben sehr hoch. Die Auswirkungen können durch günstige Einstellungen des uns ja schon bekannten `bdflush` (über procfs) eingedämmt werden.

Die große Beliebtheit von ext3 hat zwei Gründe, die ganz unabhängig von der eigentlichen Effizienz sind: Zum einen ist die Migration auf ext3 sehr einfach, da einem bestehenden ext2 lediglich ein Journal hinzugefügt werden muss, um es als ext3 einhängen zu können. Eine sauber ausgehängte („clear") ext3-Partition kann auch jederzeit als ext2 eingehängt werden. Desweiteren ist ext2 schon alt & robust. Somit wurde auch ext3 schon anfänglich als sehr stabil angesehen.

ReiserFS - ein junges, innovatives Dateisystem:

ReiserFS ist ein freies Dateisystem in der Entwicklung. Während man die Version 3 schon produktiv einsetzen kann, befindet sich Reiser4 zur Zeit noch im Test. Reiser3 bietet entweder Metadaten-Journaling (vergleichbar mit writeback bei ext3) oder Data-Journaling.

Reiser4 rühmt sich als „Atomic Filesystem". Hier herrscht grundsätzlich Data-Journaling, allerdings auf eine spezielle Art und Weise.
Das Journal/aktuelle Log hat in diesem Dateisystem keine vorbestimmte Position, vielmehr bewegt es sich möglichst sinnvoll auf der Partition. Damit wird vermieden, dass die Daten doppelt geschrieben werden müssen. Ist der Schreibvorgang komplett abgeschlossen, werden nur die Referenzen geändert - der Bereich des Journals wird in die Datei eingegliedert und ein neuer freier Bereich gewählt. Durch diese Methode bleibt gewährleistet, dass keine alten Daten unvollständig überschrieben werden. Eigene Benchmarks zeigen eine nur um 20 % erhöhte Durchschnittsdauer bei Schreiboperationen gegenübergestellt zum Schreiben ohne Data-Journaling. Das steht entgegen der ja schon vom Prinzip her erklärten etwas mehr als doppelten Dauer, die ext3 bei Data-Journaling zum Schreiben benötigt.
Diese Performanceeinbuße wird durch Techniken in ReiserFS zur Optimierung der Geschwindigkeit mehr als aufgehoben. Damit bietet Reiser4 nebst vielen anderen Vorteilen quasi Data-Journaling „für jedermann".

Allerdings birgt diese Methodik auch Nachteile. So legt ReiserFS neue Datenstrukturen zwar so an, dass Dateifragmente möglichst vermieden werden können, spätestens beim Modifizieren von Dateien entstehen aber diese. Reiser löst das mit einem sogenannten „Repacker", der ohnehin regelmäßig zum Einsatz kommen soll, um die Dancing Trees[tm] zu optimieren.
Einige Anwendungen wie z.B. Datenbanken mögen das überhaupt nicht, da ihre Suchläufe stark ausgebremst werden. Reiser4 bietet Plugins auf verschiedenen Ebenen, mit denen eine Ausnahmebehandlung solcher Anwendungen möglich wäre.

3. Zusätzliche Informationen

Migration auf ext3:

Um aus einer bestehenden ext2-Partition eine ext3-Partition zu machen, benötigt man nur das Boardtool tune2fs, um ein Journal anzulegen:

```
tune2fs -j /dev/partition
```

Auch die Art des Journalings lässt sich so festlegen:

```
tune2fs -o journal_data_ordered (journal_data_writeback, journal_data)
```

Anschließend kann die Partition als ext3 gemountet werden. Die Art des Journals kann jederzeit geändert werden.

Aktueller Status von ReiserFS:

Um ReiserFS einzusetzen, muss die Partition mit `mkfs.reiser` formatiert werden. Reiser3 ist als Journaling-Dateisystem eher weniger interessant, auch wenn es durch seine Geschwindigkeit besticht. Reiser4 ist derzeit noch in der Entwicklung bzw. Testphase. Es wird allerdings damit gerechnet, dass ca. in einem halben Jahr die nächste Release der Distribution Lindows erscheint - welche wie unlängst angekündigt Reiser4 als Standardsystem an Board hat. Man kann davon ausgehen, dass das Dateisystem bis dahin zum produktiven Einsatz geeignet ist.

Zugrundeliegende und weiterführende Literatur im Internet:

Einige der folgenden Dokumente haben auch hier nichtbehandelte, aber dennoch nicht unspannende andere Aspekte bei der Konzeption von Dateisystemen im Auge.

- The Design and Implementation of a Log-Structured File System
  von Mendel Rosenblum und John K. Ousterhout
  `http://www.cs.berkeley.edu/~brewer/cs262/LFS.pdf`
- Reiser4
  von Hans Reiser
  `http://www.namesys.com/v4/v4.html`
- Journaling Filesystems The Future Of Storage Under Linux
  Linux Magazine, August 2000, von Moshe Bar
  `http://www.linux-mag.com/2000-08/journaling_01.html`
- Journal File Systems
  Linux Gazette, von Juan I. Santos Florido
  `http://www.linuxgazette.com/issue55/florido.html`
- Journaling Filesysteme unter Linux Architektur und Funktionsweise, Verfügbarkeit und Performanz
  von Andreas Ehlers
  `http://www.fh-wedel.de/~si/seminare/ws01/Ausarbeitung/3.journalfs/`
- Filesystems HOWTO *(leider lückenhaft)*
  von Martin Hinner
  `http://www.penguin.cz/~mhi/fs/Filesystems-HOWTO/Filesystems-HOWTO.html`
- Ext3 journaling options
  EURONIA srl
  `http://www.euronia.it/includes/pdf.php?k=ARTICOLO&ID=41`

Ein guter Benchmark, der nebst ext2 als Ausgangsbasis die Performance der Journaling-Dateisysteme ext3, Reiser3, Reiser4, XFS und JFS vergleicht: `http://fsbench.netnation.com/`

Einige Hinweise:

- Meist gebräuchlich ist die Schreibweise „Journaling“, allerdings wird auch „Journalling“ verwendet, z.B. in einigen Manpages.
- Lindows.com ist trotz umstrittenen Ansehen in der Open-Source-Gemeinde zugutezuhalten, dass sie mit ReiserFS ein vielleicht für die Zukunft bedeutendes Projekt finanziell massiv unterstützen. Der Gründer von Lindows.com hat schon im Vorfeld als CEO von mp3.com großzügiges Sponsoring betrieben.
- Bei *BSD erfreut sich die Technik der „Softupdates“ größerer Beliebtheit, die dateisystemunabhängig eine interessante, wenn auch kaum gleichwertige Alternative zum Journaling darstellt. Der Text *Softupdates: eine Erklärung von Jörg Wunsch* gibt eine ganz gute Einführung ab, derzeit verfügbar unter:
  `http://mailbox.univie.ac.at/~le/softupdates.html`