



---

# Journaling

*A. Beyer*  
*Seminar Dateisysteme*  
*Thema **Journaling***  
*22.05.07*

1/37

Ein Vortrag zum Thema Journaling- Filesystems,  
teils exemplarisch an ext3,  
von den vorgestellte Konzept und Realisierungen her aber durchaus allgemeingültig.



# Gliederung

---

- **Begriffserklärung und Motivation**
- Realisierungen
- Performance
- Fazit
- Quellen & Links

In diesem Teil des Vortrags werden die Beweggründe und Motivationen zum Einführen von Journaling dargelegt.

Wiese wären zum Beispiel die erhöhte Datensicherheit, und bessere Skalierbarkeit im Bezug auf Konsistenzprüfungsdauer (fsck).



## Motivation

---

... People are building 500 gigabyte EXT2 filesystems. They take a long time to fsck. Doing a consistency check on them is a serious down time. So the real objective in EXT3 was this simple thing: **availability**.

*Zitat Dr. S. Tweedie – Ottawa Linux Symposium 2000*

Dr. Stephen Tweedie zur Vorstellung seiner Erweiterung zu ext2 um diese (unter anderem) journalingfähig zu machen.

Andere Neuerungen: H-Bäume / Online Verzeichnisgrößenänderungen (an dieser Stelle nicht von Interesse)



## Motivation

---

- Ansprüche an ein Filesystem
  - Schnelligkeit
  - Schonender Umgang mit Ressourcen
    - CPU
    - Speicherplatzausnutzung
  - Flexible Verwaltungsoptionen
  - **Konsistenz**

Ansprüche die heute an Dateisysteme gestellt werden wären wohl Schnelligkeit in Schreiben und Lesen sowohl vieler kleiner als auch sehr großer Daten.

Dabei sollte der vorhandene Speicherplatz bestmöglich ausgenutzt und die CPU geschont werden.

Plattenkontingente zu verwalten, auf verschiedenen Ebenen Zugriffskontrollen einzurichten

und große Systeme aufzusetzen sind ebenfalls Anforderungen an aktuelle Dateisysteme.

Über allem steht jedoch die Konsistenz,

da alle genannten Punkte nur nützlich sind, wenn die gelesenen Daten auch denen entsprechen die ursprünglich gespeichert wurden.



## Begriffserklärung

---

- Was ist Journaling?
  - Loggen der Meta-Daten Änderungen
  - Kommt aus Bereich der Datenbanken (Oracle / Sybase)



## Begriffserklärung

---

- Warum macht man es?
  - Konsistenz des Dateisystems
  - Kein File-System-Check mehr nötig
  - Maximieren der Uptime
  - Bessere Skalierbarkeit
    - fsck-Zeit steigt proportional zu Partitionsgröße
    - Replay-Zeit steigt proportional zu Update-Intervallen

Bzw. minimieren der DownTime die ohnehin durch das Einspielen von Updates etc. gegeben ist.

Oberstes Ziel bei Fragen der Konsistenz ist die

„Schlüssigkeit der Meta-Daten“,

nicht unbedingt die Erhaltung oder korrekte Repräsentation der Daten selbst.

fsck → nächster Vortrag

Was sind Meta-Daten? → Vortrag F. Kaplan



## Begriffserklärung

---

Wo kommt Journaling vor (chronologisch)?

- JFS von IBM seit 1990
- NTFS von MS seit 1993 (mit IBM)
- XFS von SGI seit 1994
- ReiserFS seit 2001 (ab kernel 2.4.1)
- XFS seit 2001 (ab kernel 2.6 offiziell)
- Ext3 seit 11/2001 (ab kernel 2.4.15)
- JFS seit 2002 (ab kernel 2.4.20)

JFS - Journaled File System

NTFS - New Technology File System (aus JFS entstanden)

XFS – 64-Bit-Dateisystem

2000 unter der GPL lizenziert

erschien 2001 erstmals auf Linux

ReiserFS - von Hans Reiser

zunächst nur als private Studie

erstes praxistaugliches Journaling Filesystem für Linux

EXT3 - als AddOn zu Ext2 von Dr. Stephen Tweedie

---

10 Jahre zwischen erster Realisierung und erster Implementation in Linux:

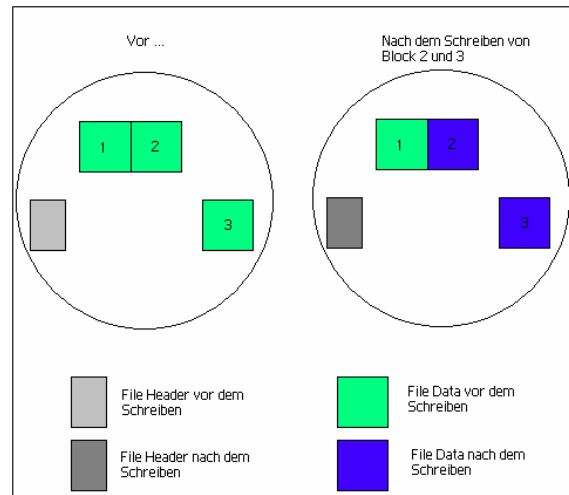
Es gab wohl in den 90ern noch keine Notwendigkeit Journaling umzusetzen, da die Partitionen noch keine kritische Größe hatten und Linux noch nicht Einzug in kommerzielle Produktivsysteme hielt.

# Begriffserklärung

Operation ohne  
Journaling auf einem  
Ext2-System

(unterbrochener  
Updatevorgang)

Metadaten konsistent  
Daten unbrauchbar



A.Beyer - Seminar Dateisysteme - Thema Journaling - 22.05.07

8/37

Beispiel:

Stromausfall nach dem Schreiben des 2. Blockes

Der neue Inhalt des Block 3 ist verloren

Block 2 hat bereits neuen Inhalt

Beim nächsten Booten wird FS-Check durchgeführt

Hierbei wird eventuelle Inkonsistenz zwischen Inode und

File gefunden bzw. repariert.

Dass die Datei nicht so vorliegt, wie gewünscht bleibt

dabei vom FS-Check unbemerkt.

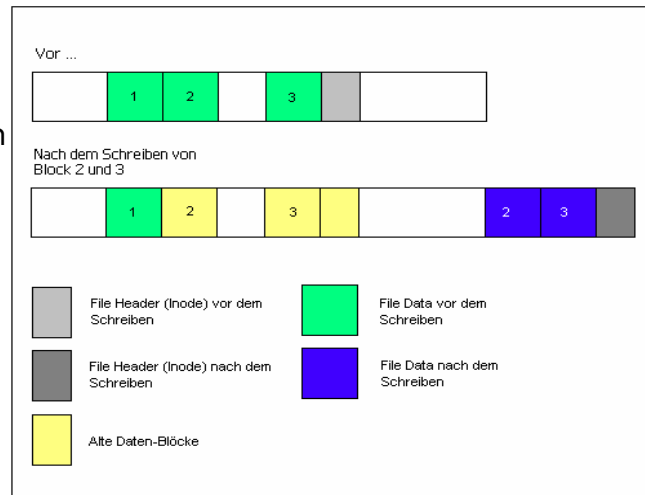
# Begriffserklärung

gleiche Operation  
auf einem  
Logging-File-System

(unterbrochener  
Updatevorgang)

Metadaten  
konsistent

Datenkopie blieb  
erhalten



A.Beyer - Seminar Dateisysteme - Thema Journaling - 22.05.07

9/37

dies wäre die Herangehensweise von XFS,  
wie man sieht sind alte und neue Daten noch vorhanden.  
Wichtiger ist der Punkt, dass auch alter und  
neuer inode noch vorhanden sind,  
was die Wiederherstellung einer konsistenten  
Verzeichnisstruktur ermöglicht.

Kein Journaling der Daten aber offensichtlich der  
Versucht diese nicht unnötig zu überschreiten  
(was, wenn überschreiten ein Löschen als Intention hatte?)  
Durch Journaleintrag Info über Konsistenz auch ohne fsck vorhanden.  
Im speziellen Beispiel sogar Option alten oder neuen  
konsistenten Zustand herzustellen  
→ jeweils mit entsprechenden gültigen Daten

# Gliederung

---

- Begriffserklärung und Motivation
- **Realisierung**
  - **theoretischer Ansatz**
  - Beispiel aus Praxis (ext3)
- Performance
- Fazit
- Quellen & Link

In diesem Teil des Vortrags werden die Mechanismen und Optionen der Realisierung eines Journal gezeigt.

(Teils allgemein, teils konkret am Beispiel ext3)



# Realisierungen

---

- Mechanismen
  - Transaktion
  - Valid-Bit
  - Journal

Allgemeiner Ansatz

Valid-Bit auf Partition

Rest von Journaling-Layer realisiert in ext3

bei anderen Systemen → stärkere Integration der Komponenten

# Realisierungen

---

- Mechanismen

- Transaktion

- ACID-Eigenschaften

- (Atomicity, Consistency, Isolation, Durability)

- einzelne zusammengehörige Operationen

- Abschluß wird mit „Commit“ bestätigt

- als ganzes konsistent

- Valid-Bit

- Journal

wichtig für uns:

-atomic

-isolated

# Realisierungen

---

- Mechanismen
  - Transaktion
  - ValidBit
    - Nach Mounten auf 1 gesetzt (dirty)
    - Nach erfolgreichem UnMounten auf 0
    - Während Mounten = 1 → Log-Replay
  
  - Journal

# Realisierungen

---

- Mechanismen
  - Transaktion
  - ValidBit
  - Journal
    - protokolliert Operationen
    - erfasst diese als Transaktion
    - wird bei Valid-Bit = 1 auf Platte angewandt

## Realisierungen

- Bsp. Transaktion → „create File“
  - Verzeichniseintrag hinzufügen
  - Verzeichnis-Timestamp ändern
  - Verzeichnis-Größe ändern
  - neue inode für File allokieren
  - Inode-Tabelle aktualisieren
  - Superblock aktualisieren
  - Inode-Map aktualisieren

Dies sind Einzeloperationen auf Meta-Daten welche das System von einem konsistenten Zustand in einen anderen überführen und deshalb zu Transaktionen zusammengefasst werden.

Sinn der Zusammenführung:

Sie werden „ganz oder gar nicht“ ausgeführt.

Da der Zustand des Systems vor und nach der Transaktion als konsistent vorausgesetzt wird und man die Transaktion in gewisser Weise atomar ausführt bleibt der Zustand konsistent.

Achtung:

Journaling bewertet diese Operationen nicht.

Sollten wieder Erwartete Befehle gegeben werden deren korrekte Ausführung die Konsistenz zerstört WIRD die Journaling-Layer diese ausführen und beim Unmount das Valid-Bit auf 0 setzen!



# Realisierungen

---

- Strategien:
  - UnDo-Journaling
  - ReDo-Journaling



# Realisierungen

---

- Strategien:
  - UnDo-Journaling
    - IST-Meta-Daten sichern
    - Meta-Daten ändern
    - bei reboot evtl. alten Zustand einspielen
    - Somit alten Zustand bewahren
    - eher verbreitet in Datenbanken
  - ReDo-Journaling

# Realisierungen

---

## ○ Strategien:

- UnDo-Journaling
- ReDo-Journaling
  - SOLL-Meta-Daten eintragen
  - Meta-Daten schreiben
  - Bei reboot diesen Vorgang wiederholen
  - Somit neuen Zustand herstellen
  - In Dateisystemen gebräuchliche Variante

Regelfall: Redo, aber jede Regel hat ihre Ausnahmen...

Es gibt keinen Königsweg, jedoch würde

UnDo bewußt neue Informationen durch alte ersetzen.

Sinn erschließt sich aus der Frage ob Datenblöcke vor oder nach Metadaten aktualisiert werden.

## Realisierungen

---

- Journaling-Stufen (in ext3 & ext4)
  - Data=journal
  - Data=ordered
  - Data=writeback

Diese Optionen beziehen sich explizit auf ext3 (und ext4)



# Realisierungen

---

- Journaling-Stufen (in ext3 & ext4)
  - Data=journal
    - Daten, sowie Meta-Daten im Journal
    - sicherste Variante
    - Vorteil: Meta-Daten / Daten passen zueinander
    - Nachteil: doppeltes schreiben
  - Data=ordered
  - Data=writeback



## Realisierungen

---

- Journaling-Stufen (in ext3 & ext4)
  - Data=journal
  - Data=ordered
    - Nur Meta-Daten im Journal
    - Daten schreiben vor Meta-Daten Aktualisierung
    - Vorteil: gültige Daten zu Meta-Daten
    - Nachteil: langsamer als „writeback“
  - Data=writeback



## Realisierungen

---

- Journaling-Stufen (in ext3 & ext4)
  - Data=journal
  - Data=ordered
  - Data=writeback
    - nur Meta-Daten im Journal
    - Daten schreiben bei fsync()
    - Vorteil: schnell
    - Nachteil: keine Datengarantie

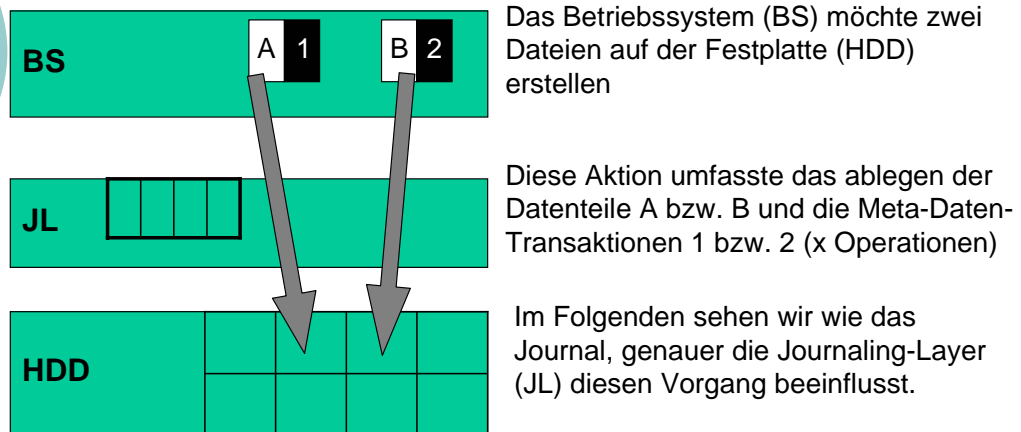
# Gliederung

---

- Begriffserklärung und Motivation
- **Realisierung**
  - theoretischer Ansatz
  - **Beispiel aus Praxis (ext3)**
- Performance
- Fazit
- Quellen & Link

In diesem Teil des Vortrags werden Optionen zum Journaling mit ext3 gezeigt.  
(full oder journal / ordered / writeback)

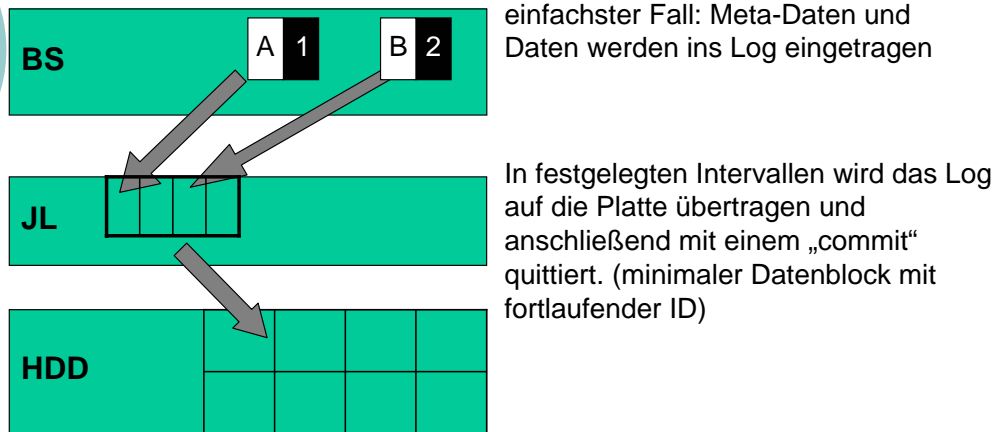
## konkrete Realisierung



24/37

Die normale Kommunikation von Betriebssystem und Festplatte läuft nun über die Journaling-Schicht, welche die Meta-Datenänderungen der gewünschten Schreiboperation zu je einer Transaktion zusammenfasst, diese bei Bedarf sammelt (aggregiert) und deren Abarbeitung protokolliert.

## 1) data = journal



25/37

genannt „Full“ (Option *data=journal*), wobei sowohl Metadaten als auch Dateiinhalte erst

ins Journal geschrieben werden, bevor sie ins Dateisystem geschrieben werden.

Dies erhöht die Zuverlässigkeit, ist aber recht langsam beim Schreiben, da alle Daten zweimal auf den Datenträger geschrieben werden müssen.

(Einmal ins Log auf Platte -> commit -> aus Log in Verzeichnisbaum

# 1) data = journal

BS

einfachster Fall: Meta-Daten und Daten werden ins Log eingetragen

JL

A 1 B 2

In festgelegten Intervallen wird das Log auf die Platte übertragen und anschließend mit einem „commit“ quittiert. (minimaler Datenblock mit fortlaufender ID)

HDD

A 1 B 2

Dort wird es abgearbeitet, bzw. in Verzeichnisbaum eingebaut.

**Konsistenz von Meta-Daten und Daten  
vor und nach  
dem Schreibvorgang.**

26/37

bitte nicht von diesem Standbild verwirren lassen.

In der Animation wird der Block **erst** auf die Platte geschrieben, committed und dann ausgeführt, bzw. kopiert.

## 1) data = ordered

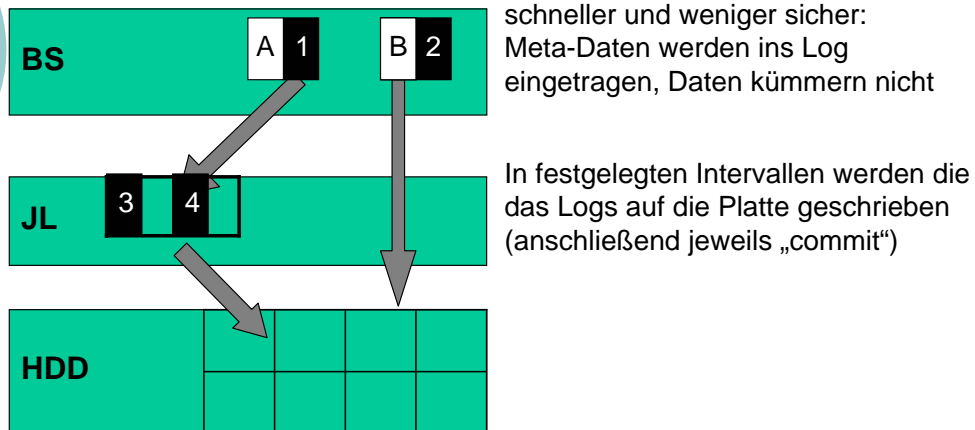


**Hohe Datensicherheit durch Gruppierung zugehöriger Daten und Meta-Daten-Blöcke, somit „geringere Verluste“ bei Absturz.**

27/37

Ordered (Option *data=ordered*) funktioniert wie Writeback. Allerdings werden Dateiinhalte (weiß) direkt ins Dateisystem geschrieben, erst danach werden die Metadaten (schwarz) aus RAM-Journal auf HDD-Journal geschrieben und committet. Dies gilt als akzeptabler Kompromiss zwischen Zuverlässigkeit und Geschwindigkeit und ist daher die Standardeinstellung.

## 1) data = writeback



**Hohe Geschwindigkeit da HDD durch schreib-/lesegünstige umgruppieren der Aufträge schneller arbeitet.  
Jedoch laufen Meta-Daten- & Daten-Kosistenz auseinander.**

28/37

Writeback (Option `data=writeback`), wobei nur Metadaten (schwarz) ins Journal geschrieben werden. Das Aktualisieren der Dateiinhalte (weiß) wird dem normalen "sync" Prozess überlassen. Dies ist wesentlich schneller, die es der Festplatte überlassen ist die Pakete geschickt zu gruppieren, birgt aber die Gefahr von Datenverlust durch abgebrochene Out-of-Order-Schreibvorgänge im Absturzfall. Dateien, die sich zu diesem Zeitpunkt im Schreibzugriff befanden, können beim nächsten Einhängen des Dateisystems an ihrem Ende Datenmüll enthalten.

# Realisierungen

## ○ Ablauf

- Schreibauftrag erteilen (BS)
- Transaktion erstellen (JL)
- Log-Speicher reservieren (JL)
- Ressourcen reservieren (JL)
- Bearbeiten der Ressource (JL)
- „Commit“ der Transaktion (JL)
- Meta-Daten schreiben (JL)
- (Daten auf Platte schreiben) (FS)

A.Beyer - Seminar Dateisysteme - Thema Journaling - 22.05.07

29/37

0.) Betriebssystem erteilt schreibauftrag als Mix aus Datenänderungen und Metadaten-Operationen

1.) JL erstellt daraus neue Transaktion mit einer eindeutigen Transaktions-ID 2.)

ist die Transaktion erstellt muss für diese Transaktion Speicher im Log reserviert werden,

um sicherzustellen, dass das Ende des Log nicht überschritten wird. (-> Deadlocks vermieden)

Speicher wird wieder freigegeben sobald die Transaktion komplett ausgeführt wurde.

3.) lock der zu ändernden Ressourcen.

4.) vermischt mit vorherigem Schritt.

5.) sind alle Änderungen durchgeführt, kann die Transaktion abgeschlossen werden.

Alle Änderungen und die entsprechenden Operationen werden in das Log geschrieben.

Da dieses Log (in-core-log) sich im Hauptspeicher befindet sind diese

Änderungen noch nicht permanent.

Dies ist erst dann der Fall, wenn dieses Log auf die Platte geschrieben wird.

Da es längere Aktionen gibt, die auf mehrere Transaktionen aufgeteilt werden,

kann bestimmt werden, ob die Ressourcen nach dem Commit einer Transaktion wieder

freigegeben werden, oder ob sie für die nächste Transaktion weiter reserviert bleiben soll.

6.) In gewissen Abständen (default 5 sec) wird das in-core-Log auf die Platte geschrieben.

An dieser Stelle wird dann eine Transaktion permanent. (-> commit)

Die Transaktion wird komplett abgearbeitet und kann erst danach wieder gelöscht werden.

7.) Geänderter Daten-Inhalt an die entsprechende Stelle auf der Platte zurückgeschrieben.



# Gliederung

---

- Begriffserklärung und Motivation
- Realisierung
- **Performance**
- Fazit
- Quellen & Links

In diesem Teil des Vortrags werden die Performance einbußen  
(und in seltenen Fällen Gewinne) durch Journaling verglichen.

# Performance

---

- **Allgemeines**

- Testumgebung: Pentium III, 16 MB RAM, 2 GB Festplatte, Linux RedHat 6.2

- **Bonnie++**

- Benchmarkprogramm: bonnie++  
(<http://www.coker.com.au/bonnie++>)

Test-Aufruf:

- `bonnie++ -d/work1 -s10 -r4 -u0`
  - `-s10` : test benötigt 10 MB
  - `-d` : im Filesystem /work1
  - `-r` : RAM - Belegung in MB
  - `-u` : der User (hier root)

Test von 2001

im Anhang finden sich neuere Tests (Okt 2006)

# Performance

## Sequential Input

Per Char: gelesen wird mit getc()  
 Block : gelesen wird mit read(2)

## Sequential Output

Per Char: geschrieben wird mit putc()  
 Block : geschrieben wird mit write(2)  
 Rewrite: gelesen mit read(2) und geschrieben mit write(2)

Benchmark - Test bonnie++													
FS	Chunk Size	Sequential Output						Sequential Input				Random Seeks	
		per Char		Block		Rewrite		per Char		Block			
		KB/sec	% CPU	KB/sec	% CPU	KB/sec	% CPU	KB/sec	% CPU	KB/sec	% CPU	#/sec	% CPU
ext2	10 MB	1471	97	14813	67	1309	14	1506	94	4889	15	309,8	10
ext3	10 MB	1366	98	2361	38	1824	22	1482	94	4935	14	317,8	10
xfs	10 MB	1206	94	9512	77	1351	33	1299	98	4779	80	229,1	11
ReiserFS	10 MB	1455	99	4253	31	2340	26	1477	93	5593	26	174,3	5

A.Beyer - Seminar Dateisysteme - Thema Journaling - 22.05.07

32/37

Zwei Werte-Gruppen werden hier angegeben.

Zum einen die Übertragungs-Geschwindigkeit (Lesen + Schreiben) des Dateisystems in KB/sec und die CPU-Belastung in %.

Je größer die Geschwindigkeit, um so besser das Dateisystem.

Genau umgekehrt ist es bei der CPU-Belastung.

Die positiv herausragenden Werte sind hier grün unterlegt, die negativen jeweils rot.

Bemerkenswert:

ext2 schreibt am schnellsten (da kein Journaling)

ext3 ist die vergleichsweise langsamste Journaling-Implementation

bis auf Reiser welches auf kleine Daten getrimmt ist lesen alle etwa gleich schnell.

# Performance

Wie man sehen kann ist das ReiserFS überlegen beim Erzeugen von Dateien. Es ist um den Faktor 10 schneller (Bereich Sequential Create, Random Create). Bei der reinen Zugriffsgeschwindigkeit ist ReiserFS nicht weit von den anderen Systemen entfernt. Zwischen diesen gibt es dann auch keine entscheidenden Unterschiede.

Benchmark - Test bonnie++													
FS	Num Files	Sequential Create						Random Create					
		Create		Read		Delete		Create		Read		Delete	
		/sec	% CPU	/sec	% CPU	/sec	% CPU	/sec	% CPU	/sec	% CPU	/sec	% CPU
ext2	16	94	99	278	99	492	97	95	99	284	100	93	41
ext3	16	89	98	274	100	458	96	93	99	288	99	97	45
xfs	16	92	99	251	96	436	98	91	99	311	99	90	41
ReiserFS	16	1307	100	8963	100	1914	99	1245	99	9316	100	1725	100

A.Beyer - Seminar Dateisysteme - Thema Journaling - 22.05.07

33/37

Wie man sehen kann ist das ReiserFS überlegen beim Erzeugen von Dateien. Es ist um den Faktor 10 schneller (Bereich Sequential Create, Random Create). Bei der reinen Zugriffsgeschwindigkeit ist ReiserFS nicht weit von den anderen Systemen entfernt. Zwischen diesen gibt es dann auch keine entscheidenden Unterschiede.

→ Hier nur create. Kein Daten-Anteil / nur Meta-Daten  
Konzepte gleich → Geschwindigkeit gleich

Im Vergleich zu vorherigem Test ist keine enge Interaktion zwischen Filesystem und Journaling-Layer nötig, daher keine Geschwindigkeitseinbußen bei der „getrennten Implementation“ der beiden in ext3

# Performance

An diesen Tabellen kann man sehen, dass ext3 bei Delete und Rename die Nase vorne hat, ReiserFS allerdings beim Create und Copy gewinnt. Außerdem fällt auf, dass Reiser bei den kleineren Files besser abschneidet

Benchmark - Test Mongo									
	Ext3fs			XFS			ReiserFS		
File-Größe	100 bytes	1.000 bytes	10.000 bytes	100 bytes	1.000 bytes	10.000 bytes	100 bytes	1.000 bytes	10.000 bytes
Create	90,07	30,68	27,13	267,86	57,94	25,99	53,05	36,38	22,27
<b>Fragm.</b>	1,32	1,38	1,44	1,02	1,01	1,02	1	1,03	1,05
Copy	239,02	75,21	55,27	744,51	149,49	55,73	126,97	84,02	43,24
<b>Fragm.</b>	1,32	1,38	1,44	1,03	1,01	1,02	1,8	1,43	1,12
Slinks	0	16,68	1,33	203,54	29,59	2,51	105,71	19,29	1,43
Read	782,75	225,74	40,51	1543,93	348,99	50,2	562,53	409,45	56,34
Stats	108,65	25,6	2,34	262,25	46,41	1,99	225,32	89,23	3,52
Rename	67,26	16,11	0,99	205,18	33,57	1,1	70,72	20,69	1,25
Delete	23,8	6,04	3,4	389,79	64,9	8,99	85,51	18,21	1,84

A.Beyer - Seminar Dateisysteme - Thema Journaling - 22.05.07

34/37

ext3 überschreibt als einziges die ausgehängten Pointer gelöschter Elemente mit Null, welches ein versehentliches Wiederherstellen der Inhalte bei Crash nach Löschen aber vor Journal-commit ausschließt.

Trotzdem gleiche Performance!

Zum Test:

delete und rename: reine Meta-Daten-Operationen -> ext3 gut

rename und vor allem copy -> Meta-Daten & Daten-Operationen -> Reiser schneller



# Gliederung

---

- Begriffserklärung und Motivation
- Realisierung
- Performance
- **Fazit**
- **Quellen & Links**

Angabe von Quellen Links und allgemeines Fazit (aus Sicht des Nutzers).

## Fazit

---

- Viel Sicherheit bei geringen Kosten
- In Produktivsystemen unverzichtbar
  - 99,9% Uptime bedeuten:  
Maximal 9 Stunden offline pro Jahr!
- Benanntes Risiko fast immer Stromausfall
  - USV-Einheit statt Journaling?

Dieses Rechnung ergibt zwar keinen Sinn,  
Wird aber ständig im Internet angeführt...  
Mal schaun wers merkt

Hinweis: Wie bereits erwähnt schützt Journaling nur vor  
der falschen/teilweisen Ausführung richtiger Befehle.  
Falsche, bzw. schädliche Operationen (aus Sicht der Konsistenz)  
werden konsequent umgesetzt.

## Quellen & Links

Linux File Systems von Moshe Bar; Verl.: McGraw-Hill (2001)  
Journaling File Systems | Linux Magazine  
<http://www.linux-mag.com/id/1180?r=s>  
Open source : JFS project Web site  
<http://jfs.sourceforge.net/>  
Dateisysteme  
<http://www.linuxfibel.de/printversion/filesys.htm>  
Linux-Grundlagen  
<http://linux.scheermann.info/content/linux.php>  
OLS Transcription Project:  
<http://olstrans.sourceforge.net/>  
ReiserFS  
<http://www.idiom.com/~beverly/reiserfs.html>  
IBM's JFS port to Linux  
<http://oss.software.ibm.com/developer/opensource/jfs/>  
SGI's XFS port to Linux  
<http://oss.sgi.com/projects/xfs/>  
LinuxUser - LinuxUser 04/2002 - Ext3fs  
<http://www.linux-user.de/ausgabe/2002/04/073-ext3/ext3.html>  
Wikipedia  
<http://de.wikipedia.org/>

A.Beyer - Seminar Dateisysteme - Thema Journaling - 22.05.07

37/37

linux-mag, linuxfibel sind tolle Quellen

oltrans bietet Mitschriebe der Präsentationen (teils mit Folien)

„Linux File Systems“ scheint in einer Art „Journaling-Gründerzeit“ geschrieben worden zu sein und stellt eher eine Hymne als ein Informationswerk dar!

(Meine Meinung... muß ja nix heißen)



# Gliederung

---

- Begriffserklärung und Motivation
- Realisierung
- Performance
- Fazit
- Quellen & Links
- **Performance aktueller Systeme**

Dies ist ein optionaler Teil zum Vergleich der doch etwas älteren Daten des vorherigen Performanceteils mit neueren Systemen. (vorher 2001 hier 10/2006)

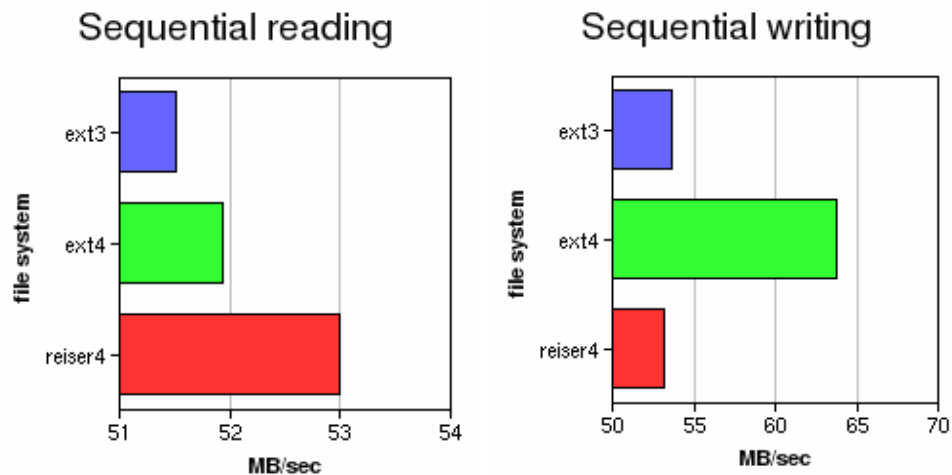
Im Vergleich stehen eine erste Version des ext4-Systems, eine frühe Version des Reiser4-Systems und deren Vergleich zum bisher gezeigten ext3.

Der Text ist aus dem englischen Original unverändert übernommen

Den Test findet man auf

[http://www.linuxinsight.com/first\\_benchmarks\\_of\\_the\\_ext4\\_file\\_system.html](http://www.linuxinsight.com/first_benchmarks_of_the_ext4_file_system.html)

## Anhang: aktuellere Tests



### Setup

A.Beyer - Seminar Dateisysteme - Thema Journaling - 22.05.07

39/37

All tests were run on a modern Intel E6600 based desktop equipped with an ATA disk.

I made a small partition exclusively for this testing.

File system under testing was recreated and freshly mounted for each and every test below, so that caching doesn't get in our way

(I could've also used [drop\\_caches](#) kernel mechanism to get rid of caches, but this is safer).

This also means that all tests were done in a pristine environment, not accounting for fragmentation and other effects happening in real environments.

ext4 was mounted with `mount /dev/hda2 /mnt -t ext4dev -o extents` so that extents are enabled.

The kernel used was 2.6.19-rc2-mm1.

The I/O scheduler was the kernel default [CFQ](#).

### Sequential reading/writing

I used an aging bonnie application to make some quick measurements of file system performance when doing sequential I/O operation.

#### Reading:

Not much surprises here. While there are some differences among the file systems, I don't think they're that much interesting.

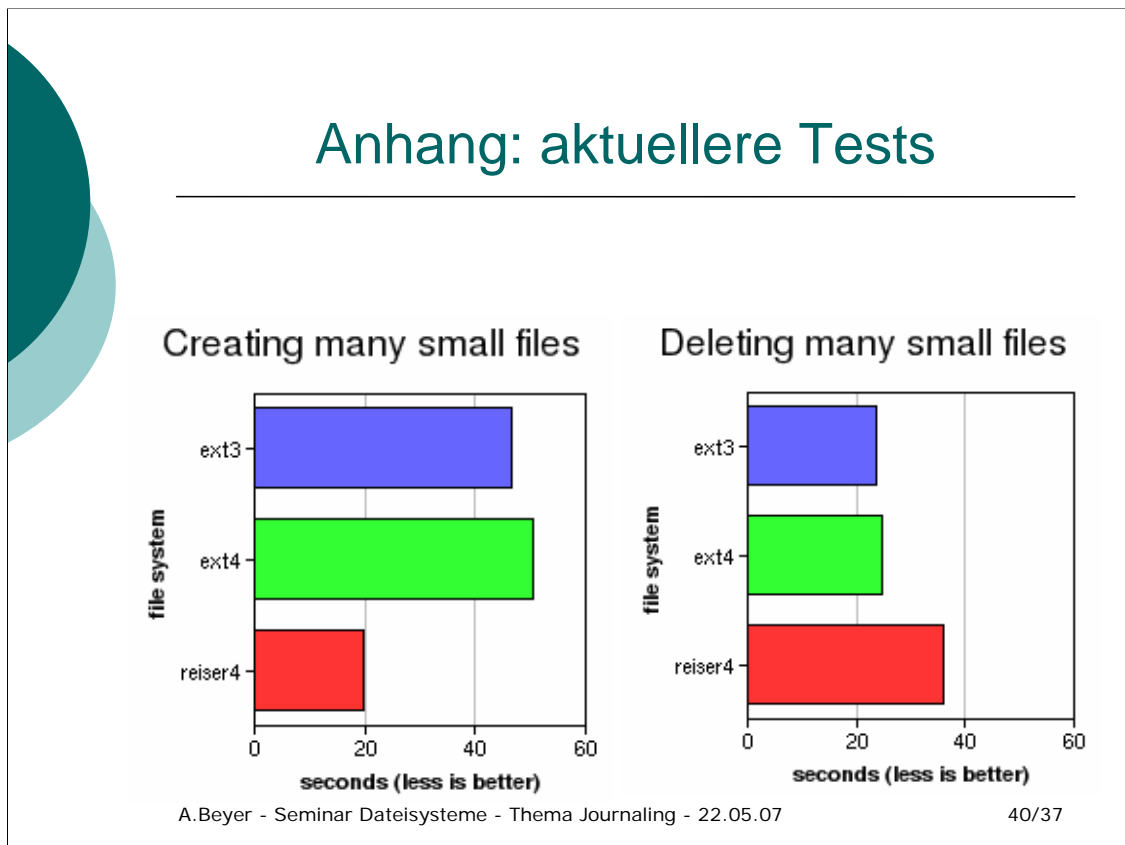
All file systems are able to read sequentially from big files with speeds close to the platter speed (57MB/s in this case).

#### Writing:

Sequential writing, on the other hand, shows that there are some real improvements built into ext4.

Probably extents together with delayed block allocation allow ext4 to come first in this benchmark, leaving the other two file systems good 20 percent behind.

## Anhang: aktuellere Tests



### Creating/deleting small files

The other typical operation file systems do in some workloads is managing many small files. To measure performance in such environment I decided to prepare a simple application (make-many-files) whose only task is to make many small files (405,000 in this case), but also distribute them in a tree like structure, so that we don't measure directory operations only (it's a known fact that the performance of file system drops rapidly when you go over some number of files in the same directory).

#### Creating:

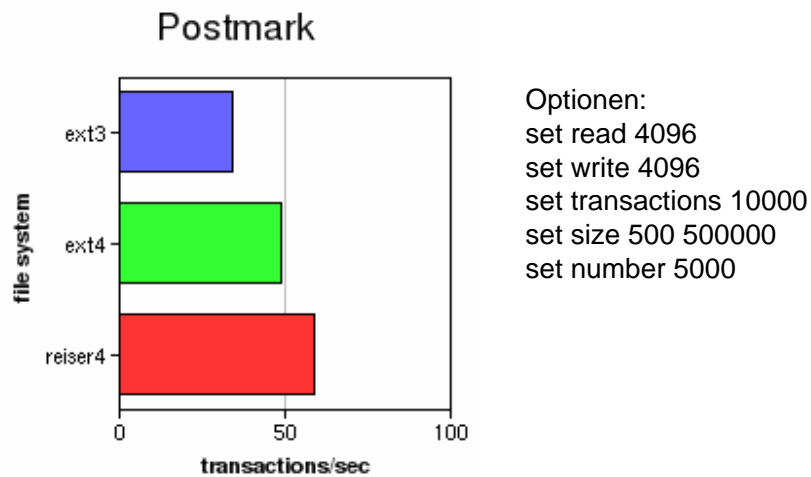
While it can be seen that ext4 shows a slight regression over ext3, the real winner here is reiser4. It seems that ext4 could use some improvements in this area (rapid file creation).

#### Deleting:

Deleting almost half a million files created in the previous step is a completely different picture. Here ext3/4 completely dominate reiser4. Once again ext4 is slightly slower, but not much. I also need to mention that for the above two tests reiser4 took much more CPU cycles.

I'm not putting the graph here because it would not be easy to interpret, but it seems that the other testers were on the right track saying that reiser4 is quite heavy on CPU under some loads.

## Anhang: aktuellere Tests



A.Beyer - Seminar Dateisysteme - Thema Journaling - 22.05.07

41/37

### Postmark

For the final test, I decided to go with a macro benchmark. I ran postmark, a benchmark that's based around small file operations similar to those used on large mail servers and news servers.

What we get out, after a few minutes of crunching with postmark, is the number of transactions per seconds. Where the bigger number indicates more performant file system. Ext4 has improved over ext3, but once again reiser4 is leading the bunch.



## Anhang: aktuellere Tests

---

- **Conclusion** The ext4 file system promises improved data integrity and performance, together with less limitations, and is definitely the step in the right way. Even if there are some regressions in our measurements, when compared to ext3, they're quite small and no doubt will be fixed before the development is finished. On the other hand, under some workloads ext4 is already showing much better results.
- Another surprise of this test is reiser4, which has the best performance in some tests.
- It should be also noted that all the file systems were completely stable during these testing, no crashes, no unexpected behaviour. So feel free to do your own tests, but still be very careful before entrusting your important data to them (except ext3, of course).