Dateisystem mit
Inline-Deduplizierung für Linux

Füllstand

Udo Seidel

Mit den wachsenden Datenmengen steigt vor allem der Anteil der redundanten Daten. Dadurch bekommt das Vermeiden solcher ungewollter Kopien einen immer höheren Stellenwert. Open-Source-Implementierungen sind hier allerdings noch recht dünn gesät. Nach SDFS stellt *iX* nun *lessfs* vor.

Dateisysteme unter Linux für die unterschiedlichsten Anwendungsfälle gibt es wie Sand am Meer. Die FUSE-Technik [a] hat die Entwicklung neuer Datenablagen sogar vereinfacht: Die Hürde der Kernel-Integration entfällt. So verwundert es nicht, dass es ein Dateisystem gibt, dessen Haupteigenschaft die Vermeidung von Daten-Dubletten ist – schließlich blockieren sie unnötig Speicherplatz und verlängern die Backup-Zeiten [1, 2].

Dem Abhilfe zu schaffen, gründete sich das *lessfs*-Projekt, dessen erste Version 0.1.15 im Frühjahr 2009 das Licht der Welt erblickte [b]. Seit November 2013 liegt das unter der GPLv3 veröffentlichte Dateisystem in der Version 1.7.0 vor. Generell empfiehlt sich bei Gebrauch ein aktueller Linux-Kernel: Erstens sind die FUSE-Implementierungen der jüngeren Vergangenheit deutlich performanter, zweitens setzen bestimmte Funktionen des Dateisystems eine bestimmte Mindestversion des Kernels voraus. Das Exportieren von *lessfs* über NFS funktioniert erst ab der Kernel-Version 2.6.30 vernünftig. Die Windows-Welt kann die Vorzüge des deduplizierenden Dateisystems über Samba nutzen.

Nicht einmal 600 kByte ist die aktuelle Version von *lessfs* schwer. Die Installation folgt dem üblichen GNU-Dreisatz: *./configure; make; make install*. Der *configure*-Schritt offenbart aber schon eine Besonderheit von *lessfs*. Es speichert die Daten nicht im Backend-Dateisystem, sondern (teilweise) in einer Datenbank. Zum gegenwärtigen Zeitpunkt kann der Benutzer zwischen Tokyo Cabinet [c], Berkeley DB [d] oder hamsterdb [e] wählen (siehe Kasten „Von Städten und Nagern“).

iX-TRACT

- Linux-Dateisysteme gibt es für alle Lebenslagen. Mit *lessfs* widmet sich ein weiteres dem Vermeiden von Datendubletten.
- Als FUSE-Dateisystem setzt es auf Standard-Dateisysteme wie ext4, Btrfs oder XFS auf.
- Ungewöhnlich ist die Architektur: Metadaten und bei Bedarf auch die Daten befinden sich in einer Datenbank.

An einem anderen Ort

Diese Design-Entscheidung der Entwickler – das Ablegen der Daten in einer Datenbank – hat gleich mehrere Konsequenzen. Sämtliche Verwaltungsaufgaben wie Sichern und Wiederherstellen oder das Bereitstellen von Plattenplatz geschehen außerhalb der von *lessfs* verwalteten Struktur. Letztere ist eigentlich nur ein leeres Verzeichnis und benötigt weder Platz noch irgendwelche leistungsverbessernden Optimierungen. Der Mountpoint ist sogar austauschbar.

Dagegen sollte der *lessfs*-Anwender die Datenbank wie einen Augapfel hüten – bekommt sie einen Kratzer ab, kann das zum totalen Datenverlust führen. Zum Konfigurieren des Dubletten-Zauberers dient eine ASCII-Datei *lessfs.cfg*, die an beliebiger Stelle liegen kann. Deshalb schreiben die *lessfs*-Verwal-

tungswerkzeuge den Pfad zur Konfigurationsdatei als Argument zwingend vor. Das ist etwas lästig – verhindert aber auch nicht, dass der Administrator sie trotzdem in */etc* hinterlegt.

Ein funktionierendes *lessfs* besteht aus vier Komponenten: der eigentlichen Software, der Konfigurationsdatei, dem Einhängepunkt und der Datenbank-Umgebung. Die letzten drei sind für jedes zu deduplizierende Verzeichnis spezifisch. *lessfs* verwendet blockbasiertes Chunking [2]. Die erforderlichen Prüfsummen ermittelt es entweder mit Tiger [f] oder SHA256 [g]; festlegen kann der Administrator das in der Konfigurationsdatei. Beim Schreibzugriff lockt es per Hash. So stellt *lessfs* sicher, dass alte Daten keine neuen überschreiben.

Hinter den Kulissen

Konfigurierbar, aber nicht dynamisch ist die Blockgröße für die Deduplizierung. Die Entwickler empfehlen, 64 oder 128 kByte zu verwenden. Bei kleineren Werten steigt der Verwaltungsaufwand für die Metadaten gewaltig und bremst das Dateisystem gehörig aus. In der Standardeinstellung verwaltet *lessfs* Daten und Metadaten in unterschiedlichen Verzeichnissen. Wer auf Geschwindigkeit optimieren möchte, setzt hier an und verwendet separate, schnelle Platten.

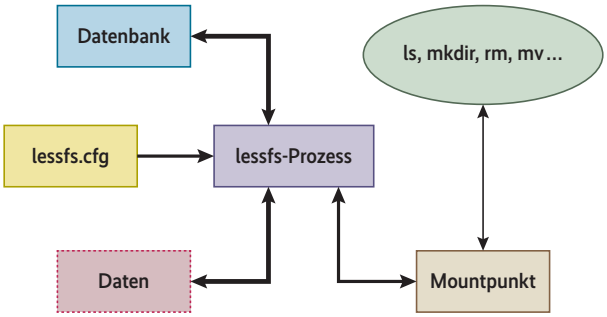
Übrigens, *lessfs* dedupliziert ausschließlich online. Es meckert sogar, wenn man ihm einen Einhängepunkt unterschieben will, in dem sich bereits Dateien befinden. Will man vorhandene Daten von unnötigen Kopien befreien, hilft nur ein Kopieren in eine von *lessfs* verwaltete Struktur. In den meisten Fällen ist der Platzgewinn größer als erwartet – schließlich bringt das Dateisystem die Fähigkeit zum Komprimieren mit und hat diese in der Standardkonfiguration eingeschaltet. Hier hat der Anwender wirklich die Qual der Wahl (siehe Tabelle „Kompressionsverfahren von *lessfs*“). *lessfs* vermerkt das verwendete Verfahren im ersten Byte des komprimierten Blocks.

Das Löschen von Daten ist keine triviale Aufgabe für *lessfs*. Zunächst verzögert es den eigentlichen Vorgang. Im Fall eines Systemabsturzes lässt sich die Datenbank auf einen konsistenten Zustand zurücksetzen. Zum Löschen verwendet es einen Hintergrundprozess, dessen Fortschritt es ebenfalls in der Datenbank protokolliert. Dadurch lässt sich der Löschvorgang beim Aushängen des Dateisystems unterbrechen und beim Wiedereinhängen fortsetzen. Das gilt übrigens analog für das „Kürzen“ (Truncation) von Dateien. Wer möchte, kann über die jeweilige *lessfs.cfg* das Verhalten aus älteren *lessfs*-Versionen einstellen; sie löschen im

Von Städten und Nagern

Lange Zeit war die Datenbank Tokyo Cabinet (TC) [c] als Grundlage für *lessfs* gesetzt. Die Entwickler waren insbesondere von der Geschwindigkeit, aber auch von der guten und gut dokumentierten API überzeugt [q]. Mit Version 1.4.0 integrierten die Entwickler Berkeley DB (BDB) [d]. Treibender Faktor war die Stabilität besonders nach Abstürzen des Systems. Tokyo Cabinet zeigt hier immer wieder Korruptionen, die letztlich Datenverlust bedeuteten. Schon mit *lessfs* 1.4.10 galt BDB als der neue Standard und TC als veraltet. Nach Aussagen des Projekts war die Performance von BDB zwar nicht überragend aber akzeptabel. Die größere Stabilität gab dann den entscheidenden Ausschlag.

Zudem ist die Benutzerbasis von BDB nicht auf *lessfs* beschränkt. Viele Anwendungen verwenden diesen Datenbank-Typ zur Datenverwaltung [r]. Dennoch ist die Zukunft von BDB bei *lessfs* limitiert. Der „neue“ Star heißt hamsterdb (HDB) [e] und ist eigentlich länger Bestandteil des Softwareprojekts – seit Version 1.3.1. Für HDB spricht die signifikant bessere Schreibperformance als BDB. Zum gegenwärtigen Zeitpunkt ist der „Nager“ für *lessfs* noch nicht stabil genug und gilt als experimentell.



Bei *lessfs* dient der Mountpoint lediglich als Schnittstelle zum Anwender. Datenanlage und Verwaltungsinformationen sind mit ihm nur über die Konfigurationsdatei verknüpft (Abb. 1).

Kompressionsverfahren von *lessfs*

Algorithmus	Indikator im 1. Byte	weitere Infos
keiner	0	–
QuickLZ (alt)	Q	*lessFS* bis Version 1.4.1; [h]
QuickLZ (neu)	R	*lessFS* ab Version 1.5.1; [h]
Snappy	S	[i]
LZO	L	[j]
GZIP	G	[k]
BZIP2	B	[l]

Listing 1

```
# ls /data/dta/
blockdata.dta
replog.dta
# ls /data/mta/
blockusage.db
__db.001
__db.002
__db.003
__db.004
DB_CONFIG
dirent.db
fileblock.db
freelist.db
hardlink.db
log.0000000001
metadata.db
symlink.db
#
```

***lessfs* verwaltet Daten und Metadaten getrennt, hier mit *file_io* Backend**

Listing 2

```
# grep ^ENC /etc/lessfs.cfg
ENCRYPT_DATA=on
ENCRYPT_META=on
# mount|grep lessfs
lessfs on /mydata type fuse.lessfs (rw,nosuid,nodev,relatime,
user_id=0,group_id=0,default_permissions,allow_other,max_read=131072)
# ls -l /mydata/
insgesamt 2
-rw-r--r-- 2 root root 7 6. Jan 18:06 datei.txt
-rw-r--r-- 2 root root 7 6. Jan 18:06 hardlink
lrwxrwxrwx 1 root root 3 6. Jan 18:13 symlink -> datei.txt
# db_dump -d a /data/mta/hardlink.db |tail
page 1: btree leaf: LSN [1][1079371]: level 1
prev: 0 next: 0 entries: 8 offset: 4000
[000] 4076 len: 16 data: 0100000000000006400000000000000
[001] 4064 len: 8 data: hardlink
[002] 4076 len: 16 data: 0100000000000006400000000000000
[003] 4020 len: 9 data: datei.txt
[004] 4052 len: 8 data: 6400000000000000
[005] 4032 len: 16 data: 0100000000000006400000000000000
[006] 4052 len: 8 data: 6400000000000000
[007] 4000 len: 16 data: 0100000000000006400000000000000
#
```

Die Verschlüsselung von *lessfs* ist selbst für durchschnittliche Sicherheitsansprüche zu harmlos.

Vordergrund und blockieren ein Aushängen des Dateisystems.

Ja, wo liegen sie denn

Wer der Arbeit des Dateisystems etwas auf die Finger schauen will, sollte einen Blick auf die Datei *<lessfs>/.lessfs/lessfs_stats* werfen. Hier vermerkt *lessfs*, wie viel Platz die Daten tatsächlich belegen: ohne Redundanzen und – falls konfiguriert – mit Komprimierung.

Wie eingangs erwähnt, liegt eine Besonderheit von *lessfs* in der Art und dem Ort der Datenablage. Verwendet der Administrator Tokyo Cabinet, hat er die Wahl, alle Informationen in die Datenbank zu schreiben oder nur die Metadaten und Verwaltungsinformationen. Bei Berkeley DB und hamsterdb ist nur Letzteres möglich. In diesem Fall kann der Benutzer zwischen zwei Speicher-Backends wählen: Der Standard nennt sich *file_io* und ist über die Variable *BLOCKDATA_IO_TYPE* in der Konfigurationsdatei festgelegt. In diesem Fall schreibt *lessfs* alle echten Daten in die per *BLOCKDATA_PATH* festgelegte Datei, also beispielsweise */data/dta/blockdata.dta*. Für die Metadaten und andere Verwaltungsinformationen schreibt *META_PATH* den Ablageort vor (siehe Listing 1).

Ein gravierender Nachteil von *file_io* besteht darin, dass die Datendatei ständig wächst. Löscht der Anwender Dateien, vermerkt *lessfs* dies zwar, gibt den Platz auf der Festplatte aber nicht frei. In der gegenwärtigen Version der Software gibt es keine Möglichkeit, diesen Mangel zu beheben. Mit dem wiederholten Kopieren und Löschen von sogar nur einer Datei lässt sich das Verhalten einfach nachstellen. Anders gesagt heißt dies auch, dass selbst ein leeres *lessfs* einen Datenträger füllen kann.

Seit Version 1.5.0 steht mit *chunk_io* ein weiteres Daten-Backend zur Verfügung. Hier organisiert das Dateisystem die Daten in einer gehashten Verzeichnisstruktur. Dabei verwendet *lessfs* dieselbe Blockgröße wie beim Identifizieren der Duplikate. Diese Art der Datenverwaltung ähnelt in gewisser Weise dem Dateisystem BTRFS [3, m] – kein Wunder, dass die Entwickler dieses als darunterliegende Struktur empfehlen. Prinzipiell funktionieren *ext4* oder XFS, harmonieren aber weniger mit der Art und Weise, in der *lessfs* die Daten ablegt. Ein großer Vorteil von *chunk_io* ist, dass das Dateisystem beim Löschen von Daten den Platz tatsächlich freigibt.

Da *lessfs* das Verzeichnis oder Dateisystem unter dem Einhängepunkt nicht verwendet, ist dieser beliebig wählbar. Alle Informationen zum Mounten muss *lessfs* in der zugehörigen Konfigurationsdatei finden, die zwingend beim Mounten anzugeben ist. Gewissermaßen entspricht sie dem Gerätenamen beim Einbinden von traditionellen Partitionen oder Volumes.

Sonst noch etwas?

Mithilfe von OpenSSL bietet *lessfs* auch eine Verschlüsselungsfunktion [n]. Leider ist der verwendete Algorithmus fest im Quelltext verdrahtet. Von der Menge der zur Verfügung stehenden Methoden haben sich die Entwickler für Blowfish [o] entschieden. Dieses Verfahren gilt bei Experten jedoch nicht mehr als ausreichend sicher [p]. Wer dem Rechnung tragen möchte, muss den Quelltext entsprechend anpassen und übersetzen. Viel eleganter wäre es gewesen, wenn der Anwender per Konfigurationsdatei

Onlinequellen

[a]	FUSE	fuse.sourceforge.net
[b]	*lessfs*	www.lessfs.com
[c]	Tokyo Cabinet	fallabs.com/tokyocabinet/
[d]	Berkeley DB Reference Guide; Introduction; What is Berkeley DB not?	doc.gnu-darwin.org/intro/dbisnot.html
[e]	hamsterdb	hamsterdb.com
[f]	Tiger	www.cs.technion.ac.il/~biham/Reports/Tiger/
[g]	SHA256	csrc.nist.gov/groups/STM/cavp/documents/shs/sha256-384-512.pdf
[h]	QuickLZ	www.quicklz.com
[i]	Snappy	code.google.com/p/snappy/
[j]	LZO	www.oberhumer.com/opensource/lzo/
[k]	GZIP	www.gzip.org
[l]	BZIP2	www.bzip.org
[m]	BTRFS	oss.oracle.com/projects/btrfs/
[n]	OpenSSL	www.openssl.org
[o]	Bruce Schneier; Blowfish Paper	www.schneier.com/paper-blowfish-fse.html
[p]	Tom Gonzalez; A Reflection Attack on Blowfish	cs.columbusstate.edu/cae-ia/StudentPapers/Y2010_TheFall/StudentPapers_CPSC6126/PaperGonzalezTom.pdf
[q]	*lessfs*-Projekt; A tale about key value databases	www.lessfs.com/wordpress/?p=378
[r]	wikipedia.de; Berkeley DB	en.wikipedia.org/wiki/Berkeley_DB

lessfs speichert die Daten nicht im eigentlichen Verzeichnis. Die _lessfs_-Struktur lässt sich an beliebiger Stelle einhängen (Abb. 2).

```
# mount|grep mydata
# lessfs /etc/lessfs.cfg /mydata/
# mount|grep mydata
lessfs on /mydata type fuse.lessfs (rw,nosuid,nodev,relatime,user_id=0,group_id=0,default_perm
issions,allow_other,max_read=131072)
# ls /mydata/
datei.txt
# cat /mydata/datei.txt
Hallo!
# umount /mydata
# ls /mydata/
#
# mkdir /mybak
# lessfs /etc/lessfs.cfg /mybak/
# mount|grep lessfs
lessfs on /mybak type fuse.lessfs (rw,nosuid,nodev,relatime,user_id=0,group_id=0,default_permi
ssions,allow_other,max_read=131072)
# ls /mybak/
datei.txt
# cat /mybak/datei.txt
Hallo!
#
```

das Verschlüsselungsverfahren festlegen könnte.

Ist die Verschlüsselung aktiviert, muss der Benutzer beim Anlegen der Dateisystem-Struktur das Password vergeben und bestätigen. *lessfs* fragt dieses dann bei jedem Mount-Vorgang ab. Verschlüsseln kann man entweder nur die Daten oder die Daten und die Metadaten. Doch selbst im zweiten Fall liegen sämtliche *lessfs*-Verwaltungsinformationen im Klartext vor und bieten eine gute Basis für Angreifer (siehe Listing 2). Unterm Strich ist die Verschlüsselung nicht wirklich praxistauglich. Vertrauliche Daten unter *lessfs* schützt der Anwender besser mit anderen Methoden.

Ein weiteres – unerwartetes – Feature von *lessfs* ist die Replikation. In der momentanen Implementierung können nur zwei Partner teilnehmen: einer fungiert als Master, der andere als Slave. Die Daten kann der Master synchron oder asynchron an den Slave senden. Dabei ist Letzteres die empfohlene Option. *lessfs* vermerkt die Veränderung auf dem Master in einer Protokolldatei. Nach dem Transfer des Protokolls auf den Slave arbeitet die dortige *lessfs*-Instanz die Informationen ab. Damit das funktioniert, muss der Benutzer entsprechende Batch-Jobs und passwortfreie SSH-Kanäle einrichten. Bei der synchronen Replikation verwendet das Dateisystem eigene Kanäle.

Bei beiden Replikationsarten ist die Empfänger-Seite nicht zum Schreiben freigegeben. So weit zur Theorie. In den Labortests hat sich das Setup als äußerst instabil und störanfällig erwiesen. Für den asynchronen Fall behilft man sich einfacher mit traditionellen File-Replikatoren. Der einzige Nachteil ist, dass der Slave noch mal die Deduplizierungsarbeit machen muss.

Fazit

Der Anfang von *lessfs* war vielversprechend. Deduplizierung und Komprimierung funktionieren tadellos. Durch das Speichern der Daten außerhalb des Dateisystems muss der Admin etwas umdenken. Zudem gibt es ausreichend Optionen zum Tunen der Performance. Bei der Verschlüsselung sollte das Projekt noch mal gewaltig Hand anlegen. Die Replikation hingegen kann eigentlich komplett wegfallen. Mehr Dokumentation insbesondere zu den Details der Architektur ist ebenfalls dringend nötig. Trotz regelmäßiger neuer Versionen ist es sehr ruhig im Umfeld von *lessfs*. Sieben Wortmeldungen in zwei Jahren auf der Mailing-Liste des Projektes sind wirklich ausgesprochen wenig. Zusammenfassend muss man sagen: Zum gegenwärtigen Zeitpunkt ist *lessfs* nicht für den produktiven Einsatz geeignet. Das Konzept zumindest verdient Beachtung. (sun)

Dr. Udo Seidel

ist studierter Mathe-Physik-Lehrer und leitet das Linux-Strategie-Team bei der Amadeus Data Processing GmbH in Erding.

Literatur

[1] Udo Seidel; Dateisysteme; Es kann nur einen geben; SDFS: Scalable Deduplicated File System; *iX* 1/2013, S. 136

[2] Nils Haustein; Massenspeicher; Weniger ist mehr; Wie Deduplizierung funktioniert; *iX* 3/2008, S. 135

[3] Udo Seidel; Dateisysteme; Datenmobile; Btrfs – das designierte Linux-Standard-Filesystem; *iX* 2/2011, S. 108

Alle Links: www.ix.de/ix1403066