Btrfs Dateisystem

Proseminar Speicher- und Dateisysteme

Tom Maier

Sommersemester 2015

Inhaltsverzeichnis

1. Einführung

Btrfs ist ein neues Copy-on-Write basiertes Dateisystem, das seit 2007 als Open Source Projekt entwickelt wird. Oracle fing an das Linux basierte Dateisystem zu entwickeln und machte Chris Mason zum Chefentwickler.
Im Laufe der Zeit wirkten auch andere bekannte Firmen an der Entwicklung mit, wie zum Beispiel Suse, Intel, Red Hat oder auch Fujitsu.

Das oftmals auch einfachheitshalber Butter-FS genannte Dateisystem durchlebte eine sehr schwierige Entwicklungsphase, geprägt durch zum Teil gravierende Fehler. So gab es eine Phase, in der das btrfs gespeicherte Daten verloren hat oder sich gar nicht mehr starten ließ. Vor diesem Hintergrund ist es kein Wunder, dass es erhebliche Diskussionen und Widerstand gab als btrfs im Jahre 2009 von der GNU General Public License lizensiert wurde und seitdem zum Linux-Kernel gehört.
Im Vergleich dazu scheiterte das ZFS-Dateisystem, welches auch auf Copy-on-Write basiert, aufgrund von Lizenzproblemen, bei der Aufnahme in den Linux-Kernel.
Erst 2014 wurde btrfs als stabil eingestuft. Die Entwicklung ist jedoch auch heute noch nicht komplett abgeschlossen, da immer noch neue Funktionen hinzukommen. Daher ist es auch kein Wunder, dass immer noch kleinere Fehler auftreten. Nutzer informieren sich darüber in einer extra angelegten Mailingliste, welche für jeden zugängig ist. So kann jeder sehen welche Probleme in der letzten Zeit aufgetreten sind und wann sie behoben wurden.

Btrfs hat den Anspruch das führende Linux basierte System zu werden und somit Ext4 abzulösen. Bereits jetzt zählt es neben Ext4 und XFS zu den bekanntesten Linux-Systemen, doch es wird wohl noch eine Weile dauern, bis es sich endgültig durchsetzen kann.

2. Copy-on-Write

Wie bereits in der Einführung beschrieben handelt es sich bei btrfs um ein Copy-on-Write Dateisystem. Im Folgenden wird kurz das wesentliche Prinzip von Copy-on-Write, auch CoW genannt, beschrieben. Später wird das Prinzip von Copy-on-Write erneut aufgegriffen und gezeigt, wie es bei btrfs umgesetzt wird.

Copy-on-Write verfolgt ein einfaches Prinzip, welches besagt „Überschreibe niemals irgendwelche Daten bzw. Metadaten.“
Somit soll CoW vor allem vor dem Verlust von Daten schützen.
Der Ausgangspunkt ist eine Datei die bearbeitet wird. Im Gegensatz zu anderen herkömmlichen Dateisystemen, werden die Blöcke, in denen die Datei gespeichert ist, nicht einfach überschrieben.

Zunächst werden die betroffenen Blöcke an einen anderen, freien Ort im System kopiert. Alle Änderungen die nun erfolgen, verändern die Kopie. Die originalen Blöcke mit der Datei bleiben während des gesamten Vorgangs in einem konsistenten Zustand. Das bedeutet, dass selbst wenn das System unvorhergesehenermaßen, zum Beispiel durch einen Stromausfall, abstürzt, kann die Originale Datei nicht verloren gehen.
Nach Abschluss aller Bearbeitungen, wird auch die Kopie in einen konsistenten Zustand überführt. Erst jetzt wird die Originale Datei aus dem System gelöscht.
Die Kopie ersetzt jetzt so zu sagen die Originalen Blöcke mit der neunen Datei.
Damit ist der Copy-on-Write Vorgang abschlossen.

3. Grundlegendes

3.1 B+-Bäumen

Btrfs verwendet eine spezielle Baumstruktur, um Datenelemente zu speichern. Es werden sogenannte CoW freundliche B+-Bäume verwendet, welche aus den B+-Bäumen abgeleitet wurden und eine Vielzahl ihrer Eigenschaften übernommen haben.

Bei B+-Bäumen werden Datenelemente lediglich in den Blättern gespeichert. Blätter sind Knoten, die keine nachfolgenden Knoten mehr haben und sich somit, bildlich gesprochen, ganz unten im Baum befinden.
Die inneren Knoten enthalten lediglich Paare von Schlüsseln und sogenannte Pointer, die auf den nächsten Knoten verweisen.
Eine weitere Eigenschaft der B+-Bäume ist, dass die Blätter in einer doppelt verketteten Liste angelegt sind. Dies verringert die Zugriffszeiten auf Daten erheblich, da es nun möglich ist über die einzelnen Blätter zu traversieren. Es ist nicht mehr notwendig einen Schritt oder gegebenenfalls mehrere Schritte zu Vorgängerknoten zu machen um die Daten eines anderen Blattes auszulesen.

Eine solche Baumstruktur, wie die der B+-Bäume, verträgt sich allerdings nicht mit dem Konzept von Copy-on-Write.
Hauptgrund sind die in einer doppelt verketteten Liste angelegten Blätter. Kopiert man Datenelemente, welche bearbeitet werden sollen, um müssen auch alle Verweise bzw. Verkettungen geändert und folglich umkopiert werden.
Bei den B+-Bäumen hätte dies den Effekt, dass große Teile, oftmals aber auch der komplette Baum umkopiert werden müsste.
Dies würde zu einer großen Ineffizienz führen, weshalb die B+-Bäume bei btrfs modifiziert wurden, damit diese besser mit dem CoW Verfahren harmonieren.

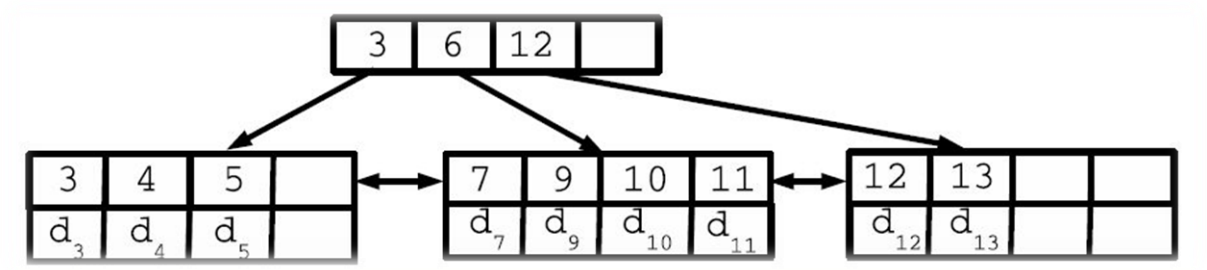

Abbildung 1: Klassischer B+-Baum

3.2 Copy-on-Write freundliche B+-Bäume

Die so genannten Copy-on-Write freundlichen B+-Bäume lassen sich im Wesentlichen auf die in 3.1 beschriebenen B+-Bäume zurückführen. Damit jene mit dem CoW Konzept vereinbar sind, wurden drei wesentliche Veränderungen vorgenommen und im btrfs umgesetzt.

Zunächst wurde die Verkettung, die zwischen den Blättern besteht, entfernt. Dadurch wurde das Fundament für ein effizient nutzbares Cow Verfahren gelegt. Das sogenannte CoWen wird erheblich vereinfacht, da durch die reduzierten Verweise weniger Knoten direkt und indirekt betroffen sind und nicht mehr umkopiert werden müssen.
Des Weiteren werden die einzelnen Knoten eines Baumes, welche jeweils einen Speicherblock symbolisieren, zusammen mit ihren Pointern von oben nach unten aktualisiert.
Wird beispielsweise eine Datei bearbeitet, verfolgt das System den Pfad von der Wurzel bis zu dem betroffen Blatt und kopiert alle Knoten bzw. Blöcke die verändert werden an einen neuen Platz. Wie in Abbildung 2b zu sehen ist, bilden die kopierten Blöcke einen neuen und unabhängigen Teil des Systems. Es werden nicht nur die jeweiligen Knoten umkopiert sondern auch die dazugehörigen Pointer. Somit müssen am Ende lediglich die Referenzen der originalen Blöcke gelöscht werden, um den Vorgang abzuschließen.

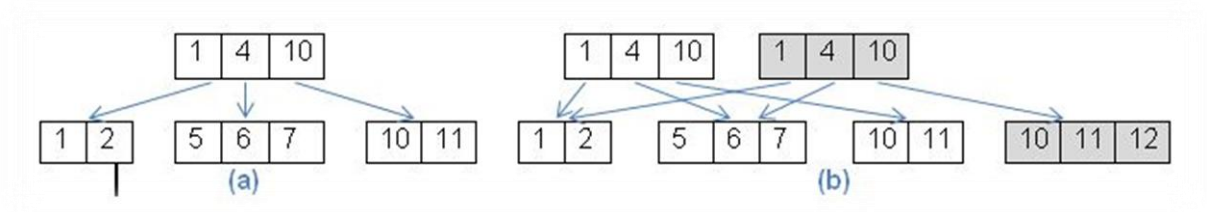

Abbildung 2: (a) stellt einen CoW freundlichen B+-Baum vor der Veränderung da; (b) zeigt einen solchen Baum während der Veränderung, wobei die grauen Blöcke umkopiert wurden und somit die veränderten Blöcke darstellen.

Als dritte wesentliche Veränderung wurde bei den CoW freundlichen B+-Bäumen in btrfs ein Referenzenzähler implementiert. Es wird für jeden Knoten gezählt, wie viele Pointer auf ihn zeigen. Hat ein Knoten keine Pointer bzw. Referenzen mehr auf sich, so ist dieser Block im System als frei gekennzeichnet.
Der Referenzenzähler ist daher im Wesentlichen dafür verantwortlich Speicherplatz zurück zu gewinnen.
Die Benutzung eines Referenzenzählers erfordert, um die Effizienz weiter auf einem hohen Niveau zu halten, einen Garbage-Collector.
Geht der Referenzenzähler eines Knoten auf null, so weiß das System, dass der Block frei ist und mit neuen Daten beschrieben werden kann. Damit die Daten aber auch wirklich von dem physischen Medium entfernt werden, wurde der Garbage-Collector zusätzlich implementiert. Dieser hat die Aufgabe die Daten des zugehörigen Blocks auf der Festplatte zu löschen, wenn auf diese keine Referenzen mehr gehalten werden.

3.3 Datenstrukturen

Das Dateisystem unterscheidet im wesentlich zwischen drei Datenstrukturen, um beispielsweise festzustellen, ob es sich um einen Zwischen- oder Blattknoten handelt.

Zum einen kennt das Dateisystem sogenannte Header. Sie stehen immer an erster Stelle und haben eine feste Größe. Ein Header besitzt bestimmte Felder, um beispielsweise Prüfsummen, IDs oder Erstellungsnummern zu speichern.
Zum anderen werden sogenannte Keys unterschieden, welche lediglich eine Objektadresse beschreiben.
Als drittes werden noch Items unterschieden, welche sich aus einem Key zusammensetzten und zusätzlich ein Offset sowie Größenfelder besitzen können.

Zwischenknoten bestehen lediglich aus einem Key und einem Pointer. Somit besitzen sie eine bestimmte Objektadresse und referenzieren auf den nächsten Knoten.
Blätter bestehen aus einem Header und jeweils einem Array für Items und einem für Datenfelder. Beide Arrays laufen aufeinander zu, so dass der freie Platz des Blocks immer zwischen den Beiden liegt.

Abbildung 3 Blattknoten mit 3 Items und zugehörigen Datenelementen

Eine Besonderheit des btrfs ist es, große Dateien sequenziell schreiben zu können und zwar mit Hilfe von Extents.
Die standartmäßige Größe eines Blocks liegt bei 4 KB. Damit größere Dateien später sequenziell auf einem Speichermedium liegen, kann das btrfs mehrere Blöcke zusammenfassen, ohne das zusätzliche Header oder Formatierungen benötigt werden.
Durch das Aneinanderreihen von einzelnen Blöcken können theoretisch beliebig große Dateien sequenziell auf die Festplatte geschrieben werden.
Wenn Daten in einem Extent gespeichert werden, wird dies im Offset des Headers vermerkt.
Kleinere Datenmengen, welche die Größe eines Blocks nicht übersteigen, werden auch weiterhin in den Blättern des Dateisystem-Baumes gespeichert.

Mit dem Gebrauch von Extents taucht ein neues Problem auf. Dadurch, dass weiterhin Copy-on-Write benutzt wird, kann es dazu kommen, dass ein sehr großes Extent kopiert werden muss.
Ein Extent, welches im Prinzip beliebig groß sein kann, umzukopieren bevor man Daten verändern kann, würde zu lange dauern und wäre nicht mehr effizient. Der Aufwand würde den Nutzen in einem solchem Fall deutlich übersteigen. Daher ist es möglich Extents zu teilen. Es werden lediglich die Blöcke des Extents kopiert, die von Veränderungen betroffen sind. Da das Dateisystem ganz genau weiß, welche Daten wo liegen, ist dieser Vorgang, der auch als Rausschneiden beschrieben werden kann, kein Problem. Später müssen die überarbeiteten Datenblöcke wieder in das Extent eingefügt werden. Somit können Extents im btrfs bearbeitet werden, ohne Effizienz abzugeben.

4. Aufbau

Das btrfs besteht aus einem Wald, dass heißt es setzt sich aus mehreren Bäumen zusammen, die zwar alle die gleiche Struktur besitzen, allerdings unterschiedliche Funktionen haben. In Abbildung 4 sieht man die verschiedenen Bäume die das btrfs bilden. In diesem Abschnitt werden die verschiedenen Bäume kurz vorgestellt.

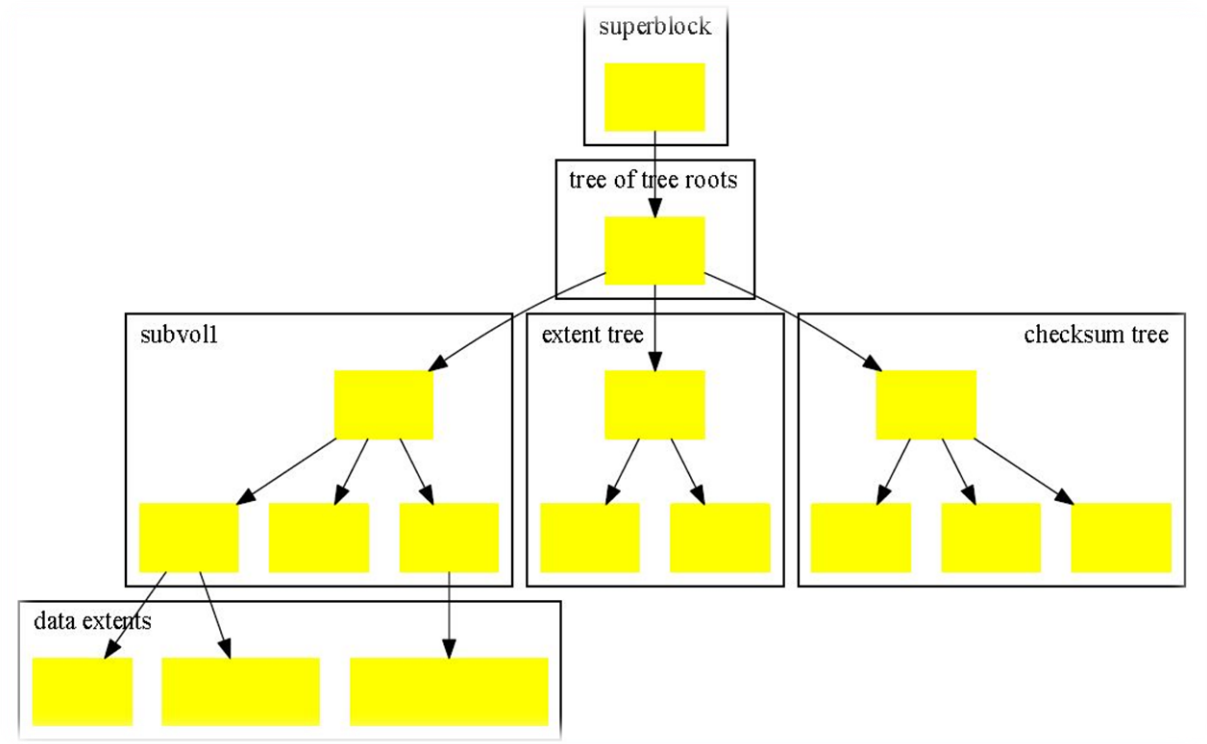

Abbildung 4 Die grobe Struktur des btrfs als Wald. „Chunk and Device Tree“ und „Reloc Tree“ nicht abgebildet, aber dennoch zugehörig.

4.1 Superblock und tree of tree roots

Die Wurzel des gesamten Dateisystems bildet der Superblock. Er besitzt, während der gesamten Lebensdauer eine feste Position und dient als Moderator zwischen all den anderen Bäumen des Dateisystems.
Der Superblock verweist auf einen weiteren Baum, welcher die Wurzeln aller anderen Bäume als Indizes speichert, die das eigentliche Dateisystem bilden.

4.2 Subvolume

Das Subvolume speichert die für den Nutzer sichtbaren Dateien und Verzeichnisse. Der Nutzer des btrfs kann sich nicht nur ein Subvolume erstellen, sondern gleich mehrere, die unabhängig voneinander sind. Es besteht aber auch die Möglichkeit Subvolumes ineinander zu verschachteln, so dass das Subvolume auf höchster Ebene die Anderen als Unterverzeichnisse hat.

4.3 „Extent Tree“

Der Extent-Tree zeichnet alle vergebenen Extents auf, indem er die Referenzen aller Zwischenknoten in umgekehrter Reihenfolge speichert. Bildlich gesprochen werden

die Pointer umgedreht und man hat nicht mehr die Richtung von der Wurzel in Blätter, sondern von den Blättern zur Wurzel.
Diese Informationen werden in einem Item im „Extent Tree“ abgelegt. Nun ist es möglich Extents beispielsweise zu bewegen.

4.4 „Checksum Tree“

Der „Checksum Tree“ speichert alle berechnet Prüfsummen. Es wird für jeden beschriebenen Block und zusätzlich auch für ein gesamtes Extent eine Prüfsumme berechnet, der Algorithmus zur Berechnung wird in 5.3 detaillierter beschrieben.

4.5 „Chunk and Device Tree“ und „Reloc Tree“

Des Weiteren gibt, es noch einen „Chunk and Device Tree“, der als Gerätemanager dient und einen „Reloc Tree“, der für die Defragmentierung verantwortlich ist.

Eine detailliertere Beschreibung dieser Bäume findet man im nächsten Abschnitt, wenn die Funktionen und Besonderheiten des btrfs beschrieben werden.

5. Funktionen

Dieser Abschnitt befasst sich mit den Funktionen des btrfs. Dabei wird der Fokus auf die wichtigsten Anwendungen beschränkt, da eine volle Funktionsbeschreibung den Rahmen dieser Arbeit übersteigen würde.

5.1 Erweiterter Speicherbereich

Das btrfs hat einen erweiterten Speicherbereich und kann somit bis zu 2^{64} Byte bzw. 16 Exbibyte groß werden. Zwar hat es damit den gleichen Speicherbereich wie das XFS Dateisystem, kann dagegen aber rund 16 mal größer werden als das konkurrierende ext4 Dateisystem. Ext4 hat im Vergleich zu btrfs und XFS lediglich einen Speicherbereich von einem Exbibyte.

5.2 Snapshots

Ein Snapshot ist ein konstantes Bild von einem bestimmten Ort zu einem bestimmten Zeitpunkt des Dateisystems.

Btrfs bietet diese Funktion an und zwar nicht nur, wie bisweilen bei anderen Dateisystemen, mit einem Lesezugriff, sondern auch mit einem Schreibzugriff.
Von den in Abschnitt 4 vorgestellten Bäumen beschränken sich die Snapshots auf das Subvolume. In btrfs ist ein Snapshot nichts mehr als Subvolume, das seine Daten und Metadaten mit anderen Subvolumes teilt.
Das Erstellen von Snapshots wird durch das Verwenden von Copy-on-Write begünstigt. Da nie die originale Datei verändert wird, sondern immer mit ihrer Kopie gearbeitet wird, muss lediglich verhindert werden, dass die originale Datei am Ende gelöscht wird. Die originale Datei bleibt bestehen. Beide Dateien, Original und Kopie, sind fortan unabhängig voneinander.

Snapshots bietet dem Anwender einige Vorteile, beispielsweise bei der Durchführung eines Backups. Soll von bestimmten Dateien ein Backup durchgeführt werden, so kann man diese zunächst snapshotten, um dann das Backup von dem Snapshot zu machen. Dadurch wird das System nicht langsamer, weil das Backup nicht an der originalen Datei, sondern an dessen Kopie im Hintergrund durchgeführt wird.
Durch den Schreibzugriff können Snapshots als Sicherungspunkte, vor beispielsweise kritischen Updates, dienen. Da sie als eigenes Subvolume funktionieren können, kann man an ihnen zur Not ein Rollback durchführen.
Allerdings findet man keine exakten Angaben über die Größe eines Snapshots in btrfs. Im Allgemeinen ist es Aufwendig herauszufinden, wie viel Speicherplatz eines Mediums aktuell belegt ist.

5.3 Prüfsummen

Btfs hat einen Algorithmus zur Berechnung von Prüfsummen implementiert. Mit Hilfe crc-32c Algorithmus werden für jedes Extent, aber auch für jeden einzelnen Block zusätzlich, die Prüfsumme berechnet und in einem Item im „Checksum Tree“ abgelegt.
Wird ein Block verändert, so wird automatisch die neue Prüfsumme ermittelt und ersetzt die alte. Verändert sich ein Extent, so wird die Prüfsumme des gesamten Extents, aber auch für jeden einzelnen Block, der zu dem jeweiligen Extent gehört, neu berechnet.

Der crc-32-c Algorithmus ist eine modifizierte, ausführlichere Version des crc-Algorithmus, arbeitet jedoch nach dem gleichen Schema. Alle crc-Algorithmen basieren auf der Polynomdivision. Sie arbeiten alle mit einer Datei, bzw. einem Block, wofür die Prüfsumme berechnet werden soll und einem Generator.
Der Binärcode des Generators besteht aus g+1 Stellen, in dem Beispiel aus Abbildung 5 also 5 +1 Stellen. An den Binärcode der Datei, für die die Prüfsumme berechnet werden soll werden g Nullen angehängt. Im gewählten Beispiel werden dann an die Datei noch 5 Nullen angehängt.

Jetzt kommt die Polynomdivision zum Einsatz, wobei die Datei durch den Generator geteilt wird. Das Ergebnis, welches man erhält ist für die Prüfsumme irrelevant.
Der Rest, der am Ende übrig bleibt, ist die Prüfsumme der Datei. Anschließend wird sie dann in dem „Checksum Tree“ abgelegt.

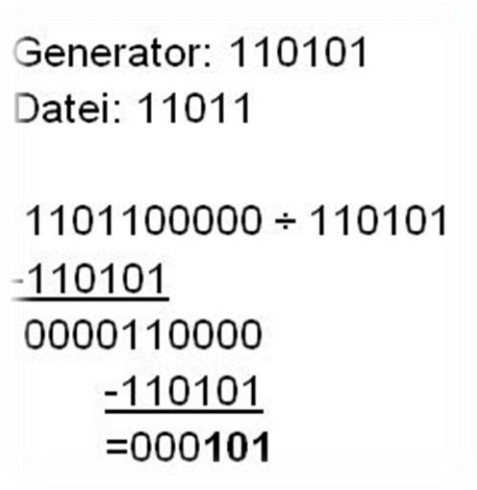

Abbildung 5 Vereinfachte Darstellung des crc-Algorithmus

Die Prüfsummen ermöglichen einen „Dateisystem scrub“, eine Anwendung, die im Hintergrund läuft, um Bitfehler zu ermitteln.
Btrfs bietet seinen Anwendern die Möglichkeit im Hintergrund nach Binärfehlern zu suchen und greift dabei auf die Prüfsummen zurück. Beim sogenannten „scrubbing“ wird die Prüfsumme für einen Block erneut ausgerechnet und mit der verglichen, die im „Checksum Tree“ gespeichert ist.
Stimmen die beiden Prüfsummen überein, ist die Datei nicht beschädigt. Gibt es allerdings eine Abweichung zwischen den beiden Prüfsummen, so liegt ein Bitfehler vor.
Das System kann nun versuchen diesen automatisch zu beheben. Dies ist in der Regel kein Problem, wenn der Nutzer beispielsweise mit einem RAID Level 1 arbeitet. In diesem speziellen Fall liegen alle Dateien in doppelter Ausführung vor. Die beschädigte Datei kann nun durch ihren „Doppelgänger“ ersetzt werden, sofern sie nicht auch beschädigt ist.

In Zukunft sollen auch noch alternative Algorithmen zur Verfügung stehen, um Prüfsummen auch auf anderen Wegen berechnen zu können. Genaue Algorithmen sind derzeit allerdings noch nicht bekannt.

5.4 Datenkomprimierung

Btrfs hat derzeit zwei Algorithmen implementiert, mit denen man Dateien komprimieren kann.
Zum einen können Nutzer ZLIB nutzen um ihre Daten zu komprimieren. Dieser hat einen vergleichsweise hohen Komprimierungslevel, arbeitet dagegen allerding sehr langsam.
Zum anderen können die Nutzer auf LZO zurückgreifen um ihre Daten zu komprimieren. Den Vorteil den LZO mit sich bringt ist seine Schnelligkeit Daten zu komprimieren, aber auch zu extrahieren. Dafür werden die Daten nicht so klein wie mit ZLIB.
Für den alltäglichen Gebrauch eignet sich LZO mehr, da die Daten schnell wieder extrahiert sind und ZLIB zu langsam arbeitet und sich eher für Daten lohnt, die man seltener braucht.

In absehbarer Zeit sollen dann noch weiter Algorithmen hinzukommen, mit denen man seine Daten noch besser komprimieren kann. Derzeit ist die Einführung von LZ4 geplant, als Kompromiss zwischen ZLIB und LZO. Des Weiteren soll snappy implementiert werden.

5.5 Volumenverwaltung

Um Speichermedien zu verwalten greift btrfs nicht auf die standartmäßigen Gerätemanager von Linux, wie LVM oder mdadm, zurück. Dafür wird der „Chunk and Device Tree“, in 4.5 schon kurz vorgestellt, verwendet.
Schließt man ein Speichermedium an, so wird es von btrfs zunächst in große Stücke aufgeteilt, auch Chunks genannt.
Es gibt zwei Arten von Chunks zum einen die physischen, in die ein Speichermedium aufgeteilt wird, und zum anderen logische Chunks.
Logische Chunks beinhalten die eigentlichen Daten, dass bedeutet das verschiedene Knoten mit ihren Daten zu einem Chunk/Stück zusammengefasst werden, bevor sie endgültig auf ein Medium übertragen werden.
Der „Chunk Tree“ kümmert sich dabei um die Zuordnung von logischen zu physischen Chunks, also die eigentliche Übertragung von den Daten auf beispielsweise eine Festplatte. Der „Device Tree“ kümmert sich um die umgekehrte Zuordnung.

Btrfs unterstützt verschiedene RAID Level, nach denen die physischen Chunks aufgeteilt werden. Derzeit sind die Level 0, 1, 5, 6 und 10 möglich.
Verwendet ein Nutzer RAID 1 so werden die physischen Chunks in zwei Gruppen aufgeteilt, da alle Daten doppelt gespeichert werden.
Damit Daten gleichmäßig auf alle angeschlossenen Geräte verteilt werden gibt es den Balance Algorithmus, der in Abbildung 6 veranschaulicht wird.

.

(a) 2 disks	logical chunks	disk 1	disk 2	
	L_1	C_{11}	C_{21}	
	L_2	C_{12}	C_{22}	
	L_3	C_{13}	C_{23}	
(b) disk added				disk 3
	L_1	C_{11}	C_{21}	
	L_2	C_{12}	C_{22}	
	L_3	C_{13}	C_{23}	
(c) rebalance				
	L_1	C_{11}	C_{21}	
	L_2		C_{22}	C_{12}
	L_3	C_{13}		C_{23}

Abbildung 6 Der Balance Algorithmus

Bei (a) sieht man, dass die drei logischen Chunks (L_1 – L_3) gleichmäßig auf die zwei CDs verteilt wurden. Jeder logische Chunk wurde einmal auf CD 1 und einmal auf CD2 gespeichert.
Bei (b) wird nun eine dritte CD hinzugefügt. Die Daten sind nun nicht mehr gleichmäßig verteilt, da CD 3 im Monet noch keine Daten besitzt.
Der Nutzer hat jetzt die Möglichkeit den Balance-Algorithmus zu verwenden, um ein Gleichgewicht zwischen den drei CDs herzustellen, wie in (c).
Nach dem ausbalancieren, hält jede CD zwei logische Chunks und jeder logische Chunk liegt auf zwei verschiedenen CDs. Somit ist das Gleichgewicht wieder hergestellt.
Der Algorithmus funktioniert auch andersherum, wenn man eine CD wieder entfernen würde. Die Daten würden dann von der jeweiligen CD entfernt werden und auf die übrigen verschoben werden, sodass am Ende wieder ein Gleichgewicht besteht.

Dadurch haben die Nutzer von btrfs gewisse Vorteile. Raid Level können sehr leicht geändert werden, da der Balance-Algorithmus die neue Verteilung der Daten übernimmt.
Verschiedene Subvolumes können verschiedene Raid Level besitzen, wobei Metadaten eigentlich standartmäßig RAID Level 1 besitzen.
Außerdem ist es jetzt möglich Geräte zur Laufzeit hinzuzufügen oder zu entfernen.

5.6 Defragmentierung

Je öfter Dateien bearbeitet werden und somit durch Copy-on-Write an andere Plätze im System geschrieben werden, desto mehr verlieren sie die Verbindung zu ihren alten Snapshots.

Um unteranderem diesem Effekt entgegen zu wirken, muss das System defragmentiert werden. Dafür wurde der Reloc-Algorithmus im brtfs implementiert, der ebenfalls Copy-on-Write benutzt.
Wenn ein Extent bewegt werden muss, wird es zunächst an seinen zukünftigen Ort kopiert.
Im nächsten Schritt werden alle Knoten ermittelt, die direkt oder indirekt Referenzen auf das Extent halten.
Nun werden alle Bäume gegebenenfalls auch mehrere Subvolumes geklont indem man als zweite Wurzel den Reloc-Baum ansetzt.
Im letzten Schritt müssen der Reloc-Baum und der Originale wieder zusammengeführt werden. Dafür werden beide Bäume von oben nach unten traversiert. Wenn man dabei einen Knoten im Reloc-Baum findet, der sich von seinem Knoten an gleicher Stelle im originalen Baum unterscheidet, so werden die beiden einfach getauscht.
Am Ende erhält man den originalen Baum mit allen neuen Referenzen für das Extent. Der Reloc-Baum wird verworfen.

6. Zusammenfassung

Das btrfs Dateisystem hatte eine schwierige Entwicklungsphase, musste immer wieder Rückschläge hinnehmen und hatte folglich lange mit Stabilitätsproblemen zu kämpfen. Aktuell treten immer noch kleinere Probleme auf, was daher kommt, dass die Entwicklung nicht komplett abgeschlossen ist und sich die Entwickler auch noch einiges vorgenommen haben. So soll beispielsweise ein schneller offline Dateisystemcheck sowie ein online Dateisystemcheck kommen.
Dennoch gilt btrfs im Moment als stabil und kann benutzt werden.
Btrfs will durch seinen vielen neuen Features überzeugen, vor allem durch Snapshots und den Gebrauch von Extents und ist derzeit auch das einzige Dateisystem im Linux-Kernel, das Copy-on-Write anbietet.
Blickt man auf die Performance hebt sich das btrfs nicht von seinen stärksten Konkurrenten (ext4 und ZFS) ab. Alle drei liegen unterm Strich ungefähr auf demselben Level.
Dafür verspricht btrfs, durch den Einsatz von Copy-on-Write, eine höhere Datensicherheit als die Anderen.
Abschließend kann man sagen, dass btrfs vieles anders und auch neu macht als andere Dateisysteme und sich somit abhebt. Auch wenn btrfs auf einem guten Weg ist ext4 an der Spitze der Linux Systeme abzulösen, hängt es aufgrund kleinerer Probleme zurück.

Somit kann man sagen, dass btrfs noch nicht die Gegenwart gehört, es aber sicherlich das Dateisystem der Zukunft sein wird.

7. Quellen

- „Main Page“, zuletzt aktualisiert am03.08.2015, https://btrfs.wiki.kernel.org/index.php/Main_Page,
- Ohad Rodeh, Josef Bacik, Chris Mason, „BTRFS: The Linux B-Tree Filesystem“, erschienen am 09.06.1012, http://domino.research.ibm.com/library/cyberdig.nsf/papers/6E1C5B6A1B6EDD9885257A38006B6130/$File/rj10501.pdf
- „Schnappschuss (Informationstechnik), zuletzt aktualisiert am 19.09.2015 https://de.m.wikipedia.org/wiki/Schnappschuss_(Informationstechnik)
- „Btrfs“, zuletzt aktualisiert am 15.08.2015, https://en.m.wikipedia.org/wiki/Btrfs
- Jan Kára, “Ext4, btrfs, and the others” http://atrey.karlin.mff.cuni.cz/~jack/papers/lk2009-ext4-btrfs.pdf
- „Fehlererkennung mit CRC“, zuletzt aktualisiert am 28.04.2013 http://banane-krumm.de/crc/index.html
- „Zyklische Redundanzprüfung“, zuletzt aktualisiert am 04.08.2015, https://de.m.wikipedia.org/wiki/Zyklische_Redundanzprüfung
- Compression“, zuletzt aktualisiert am 15.07.2015, https://btrfs.wiki.kernel.org/index.php/Compression
- Kofler, Michael: Linux das umfassende Handbuch, Bonn 2014, S.840-852
- „ext4”, zuletzt aktualisiert am 03.08.2015, https://en.wikipedia.org/wiki/Ext4