Dateisystem Benchmark: Btrfs & F2FS

Sven Fiergolla & Tobias Dahlem
Betriebssysteme WiSe 2019/2020
Universität Trier

3. März 2020

1 Flashspeicher

Flashspeicher werden heutzutage in vielen Bereichen verwendet, wie Computern, Smartphones und Servern. Neben ihren Vorteilen, wie die schnellere Geschwindigkeit, gibt es aber auch einige Einschränkungen, wie das Bedürfnis die Zellen zu löschen bevor sie beschrieben werden oder dass die einzelnen Blöcke nur eine beschränkte Lebensdauer besitzen. In dem Bereich der Dateisysteme gibt es verschiedene Vertreter, welche die Eigenschaften der Flashspeicher berücksichtigen. Zu den durch die Bauart bestimmten Charakteristika zählen das Adress Mapping, die Garbage Collection und das Wear Leveling [2].

Im Rahmen des Adress-Mappings erfolgt die zuordnung der logischen Adressen der Speicherblöcken zu den physischen. Durch die Besonderheit des Flashspeichers, wie zum Beispiel dem Bedürfnis eine Zelle zu löschen bevor sie geschrieben werden kann, kann dieses Mapping häufig geändert werden. Die Besonderheit der Garbage Collection ist, dass je nach Granularität in den löschbaren Einheit der Speicherblöcke mehrere Dateien gespeichert sein können. Nach der Änderung eines Teils der Daten aus einem Block, können also bestimmte Teile veraltet sein. Im Rahmen der Garbage Collection liegt also der Fokus darauf, veraltete Daten effizient auszusortieren ohne die I/O-Performance zu verschlechtern, um nicht unnötig Speicherplatz zu verbrauchen. Im Rahmen des Wear-Levelings wird versucht den Verschleiß aller Speicherzellen auf einem gleichmäßigem Stand zu halten, bedingt durch die beschränkte Haltbarkeit dieser Zellen [1].

Die meisten Dateisysteme werden dabei nicht direkt auf Flashspeicher zugeschnitten und führen diese Aufgaben nicht selbst aus, sondern verwenden dafür als Schnittstelle den sogenannten File Translation Layer (FTL) der die zuvor erwähnten Aufgaben umsetzt und dem Dateisystem eine definierte Schnittstelle zur Verfügung stellt ohne den Zugriff auf die „raw“ Speicherzellen zu gewähren [2]. Im folgenden Stellen wir zwei aktuelle Dateisysteme vor und Evaluieren ihre Performance in einem Benchmark-Test. Zu den Dateisystemen zählen das Flash-Dateisystem F2FS und dem Copy-On-Write Dateisystem Btrfs.

2 Btrfs

Btrfs (B-tree FS) ist ein Copy-On-Write-Dateisystem (CoW), das seit 2007 für Linux und seit 2016, mit WinBtrfs für Windows 7 entwickelt wird. Die Kernstruktur von Btrfs ist ein Copy-on-write B-Baum und wurde ursprünglich von dem IBM-Forscher Ohad Rodeh auf der USENIX 2007 vorgestellt [4]. Zusammen mit Chris Mason [3], ein ehemaliger ReiserFS Entwickler half bei der Arbeit an dem Dateisystem welches fast ausschließlich solche B-Bäume verwendet, für Meta- und Nutzdateien sowie rekursiv zur Verfolgung der Speicherzuteilung der Bäume selber[5]. Seit 2009 ist Btrfs teil des Linux-Kernel und ist das Standard Dateisystem von MeeGo und sollte seit

mehreren Versionen auch in Fedora als Standard Dateisystem verwendet werden, jedoch gab Red Hat 2017 bekannt, Btrfs nicht mehr zu unterstützen und langfristig zu entfernen.

Btrfs weist zahlreiche Gemeinsamkeiten mit ZFS auf, welches jedoch wegen seines Lizenzstatus für die Verwendung im Linux Kernel ungeeignet war. Beide Dyteisysteme basieren auf *Copy-On-Write*, wobei eine Kopie erst dann angefertigt wird, wenn sie wird. Zudem implementieren sie beide, Volume-Management, Schutz vor Datenübertragungsfehlern durch Prüfsummen und integriertes RAID. Btrfs baut mit dem B-Baum auf einem zentralen Konzept von XFS oder Reiser4 auf. Mit btrfs-convert können bestehende ext3- und ext4- Dateisysteme in ein Btrfs Dateisystem konvertiert werden, sowie umgekehrt. Weitere Features sind der erweiterte Speicherbereich (264 Byte), effizientes Speichern kleiner Dateien und Verzeichnisse, dynamische Inodes Snapshots, Subvolumes und iDatenkompression auf Dateisystemebene.

Das von btrfs implementierte RAID bietet gegenüber klassischen Hardware- oder Software-RAID Implementierungen den Vorteil, dass zwischen belegten und freien Datenblöcken unterschieden werden kann und somit bei der Rekonstruktion eines gespiegelten RAID-Volumens nur belegter Plattenplatz gespiegelt werden muss. Im Gegensatz zu klassischen RAID-Verfahren wird mit Hilfe von größeren Datenblöcken gearbeitet und es erfolgt beispielsweise im RAID 1 keine Spiegelung der Datenträger, sondern es wird sichergestellt, dass jeder Datenbereich auf wenigstens zwei Datenträgern abgelegt wird. So wird es möglich einen RAID 1 aus einer ungeraden Anzahl von Datenträgern unterschiedlicher Kapazität, unter voller Ausnutzung derer Kapazität, zu bilden. Kommt es unter Nutzung des RAID 1 zu „Bitrot“ oder sonstigem schleichenden Verlust von Informationen, fällt dies dank dem Einsatz von Prüfsummen beim folgenden Zugriff auf und der Fehlerhafte Block wird verworfen und durch den gespiegelten Datenbereich eretzt. Btrfs bietet ebenfalls eine RAID 0 Implementierung, welche für sich für den Einsatz von nicht kritischen Daten eignet, da dennoch mehrere Disks zu einem Eigenständigen logischen Volume zusammengefügt werden können für sehr größe Dateien. Zudem bieten sie eine deutlich gesteigerte Geschwindigkeit, vorallem beim sequentiellen Lesen, jedoch auch bei den wahlfreien Zugriffen wie in Abbildung 4 ersichtlich ist.

3 F2FS

F2FS ist ein Dateisystem speziell nur für Flashspeicher und wurde von Samsung 2012 veröffentlicht. Entwickelt wurde das Dateisystem mit dem Ziel der Optimierung von Performance und Lebenszeit von Flashspeichern, indem die speziellen Charakteristika berücksichtigt wurden. Dabei wurde das System für Linux als Open-Source Projekt entwickelt und ist seit dem Kernel 3.8 verwendbar, klassische Anwendungsbereiche stellen SD-Karten, embedded multimedia cards (eMMC) und klassische SSDs dar. Unter dem Android-System findet es zum Beispiel Verwendung in Smartphons von Huawei und dem Galaxy Note 10 von Samsung. Im Vergleich zu anderen Flash-Dateisystemen, kann F2FS allerdings nicht direkt auf den ,,raw"-Speicherzellen arbeiten, sondern nutzt dafür den FTL des Speichers. Im Gesamten verfügt F2FS über viele Möglichkeiten das System an die benötigt Einsatzumgebung anzupassen. Grundlegend wird dabei das UNIX-Prinzip der Inodes und Datenblöcke zur Speicherung verwendet, zusätzlich wird dabei noch der Ansatz eines Log-structured File Systems (LFS), im Sinne eines append-only loggings, verfolgt [2].

Im Folgenden werden einige Besonderheiten des F2FS Dateisystems aufgezeigt, welche das System charakterisieren, dazu zählen das On-Disk-Layout, das Multi-Head Logging, die Indexstruktur und die Garbage Collection.

Das On-Disk-Layout orientiert sich von der Blockgröße an der Größe von 4KB. Im Gesamten werden aller Speicherzellen in Blöcke dieser Größe unterteilt. Die weiteren übergeordneten Einheiten sind Segmente, Sections und Zonen. Im Gesamten ist das Volumen in 2 Bereiche aufgeteilt, die Metadaten und der Hauptbereich. Zu den Metadaten zählt der Superblock mit allgemeinen Informationen über das Dateisystem. Die Checkpoints geben im Rahmen des LFS validen Zustände an. Die Segment-Information-Table wird für jedes Segment gespeichert, welche beinhalteten Blöcke noch gültig sind oder ob er im Rahmen der Garbage Collection gereinigt werden könnte. In der Segment Summary Area werden sogenannte Informationen für alle Blöcke eines Segments gesichert, sie beinhalten Informationen wie zu welcher Datei jeder Block gehört. Die Node-Address-Table ist eine Block-Adressen-Tabelle, welche für die physischen Adressen jedes Speicherblocks eine logischen Adresse kennt. Durch diese Tabelle wird das sogenannte wandernde Baum Problem, welches beim Ändern von Daten entsteht, bekämpft. Verweise in Blöcken zeigen in F2FS daher nicht auf die physischen Blöcke, sondern speichern eine logische Adresse und erfragen die physische über die NAT. So müssen bei der Adressänderung eines Blocks nicht mehr alle Verweise angepasst werden, was einen hohen Schreibaufwand bedeutet, sondern nur die Zuordnung innerhalb der NAT. Zusätzlich wird der Metadaten-Teil nicht im Sinne eines LFS verwaltet, sondern In-Place im Rahmen einer Schattenkopie. Um den Schreibaufwand bei diesen Random-Writes auf bestimmten Speicherzellen nicht zu überlasten, werden häufig geschriebene Daten wie die NAT in einen flüchtigen Speicher gehalten und nur zum setzten eines Checkpoints auf die veraltete Schattenkopie geschrieben [2].

Im Hauptspeicher sind die iNodes und Datenblöcke abgelegt, dabei wird hier im Sinne eines LFS vorgegangen. Allerdings liegen diese in verschiedenen Bereichen, denn F2FS verwendet den Ansatz eines Multi-Head Loggings und hat sechs parallele Logsegmente, deren parallel Verwendung durch die klassische Flash-Architektur möglich ist. Die Idee dabei ist, die Daten, iNodes und Datenblöcke, in Klassen hot/warm/cold zu klassifizieren, in Bezug auf ihre Update-Frequenz. Dadurch soll ein häufiges neu schreiben von validen Teilsegmenten vermieden werden, wenn nur Teil des Segments veraltet sind. Ebenfalls wird dadurch die Anzahl der veralteten Daten reduziert und die Garbage Collection vereinfacht [2].

Die Garbage Collection (GC) wird auf zwei Arten durchgeführt, On-Demand und im Hintergrund. On-Demand wird dabei verwendet, wenn nicht genügend Speicherplatz zum schreiben verfügbar ist. In diesem Sinne wird dann, mit einem Greedy Algorithmus, auf Basis der wenigsten gültigen Blöcken ein Opfersegment ausgesucht. Im Hintergrund wird die GC mit einen Kosten-Effizienten Algorithmus bei einer geringen Auslastung des Dateisystems automatisch ausgeführt.

Dieser wählt Opfersegmente auf Grund ihres Alters und der Anzahl der gültigen Blöcke aus. Die Blöcke, welche noch gültig sind, werden dann wie bei einem normalen Schreiben wieder in die aktiven Segmente eingereiht und im Anschluss geschrieben. Die Besonderheit hierbei ist, dass wieder eine neue bessere Klassifizierung des hot/warm/cold-Schemas vorgenommen werden kann, da nun mehr Informationen über die Blöcke vorhanden sind [2].

4 Benchmarks

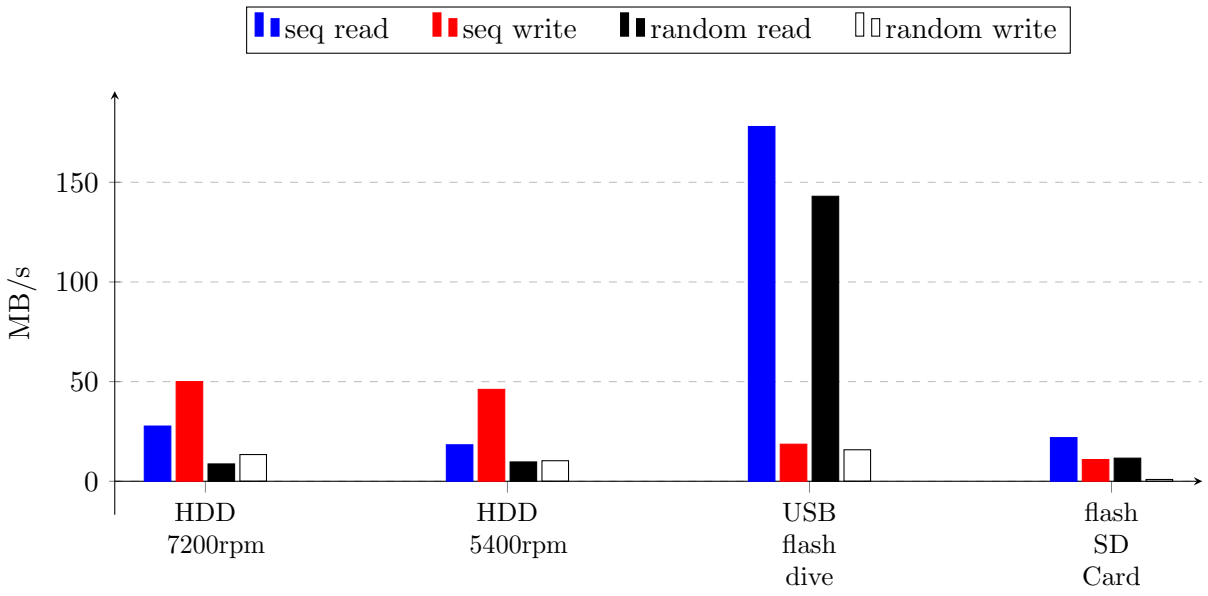

Abbildung 1: F2FS raw performance

Literatur

[1] Tae-Sun Chung, Dong-Joo Park, Sangwon Park, Dong-Ho Lee, Sang-Won Lee, and Ha-Joo Song. A survey of flash translation layer. *Journal of Systems Architecture*, 55(5):332 – 343, 2009.

[2] Changman Lee, Dongho Sim, Jooyoung Hwang, and Sangyeun Cho. F2fs: A new file system for flash storage. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pages 273–286, 2015.

[3] Chris Mason. The btrfs filesystem. *The Orcale cooperation*, 2007.

[4] Ohad Rodeh. B-trees, shadowing, and clones. *ACM Transactions on Storage (TOS)*, 3(4):1–27, 2008.

[5] Ohad Rodeh, Josef Bacik, and Chris Mason. Btrfs: The linux b-tree filesystem. *ACM Transactions on Storage (TOS)*, 9(3):1–32, 2013.

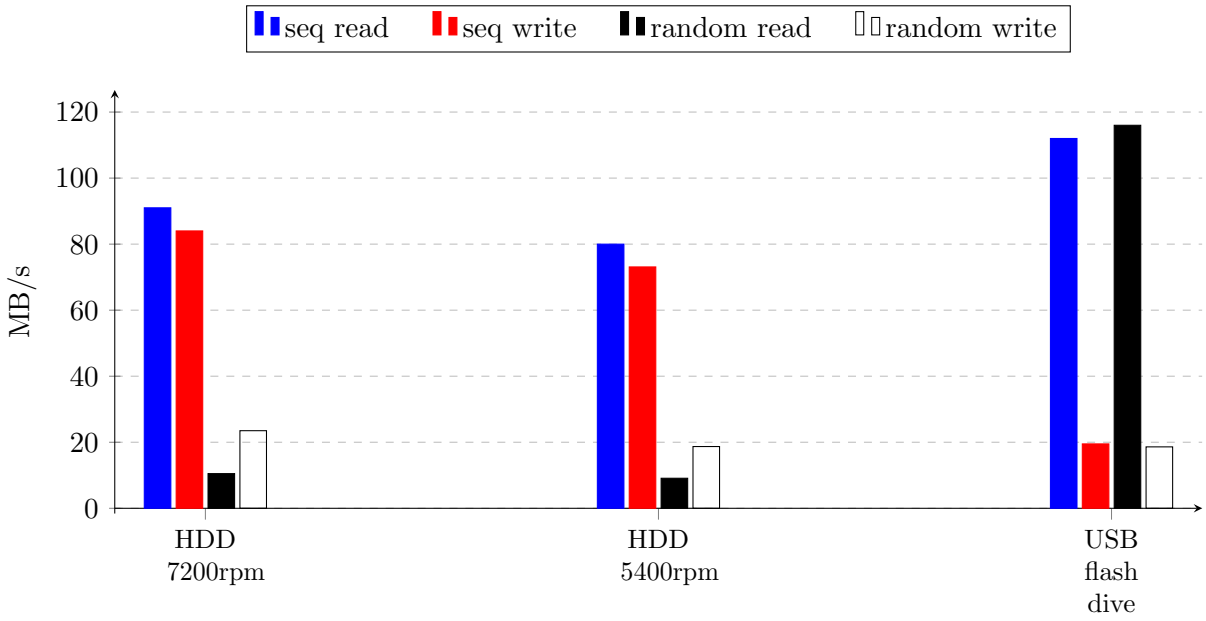

Abbildung 2: BtrFS raw performance

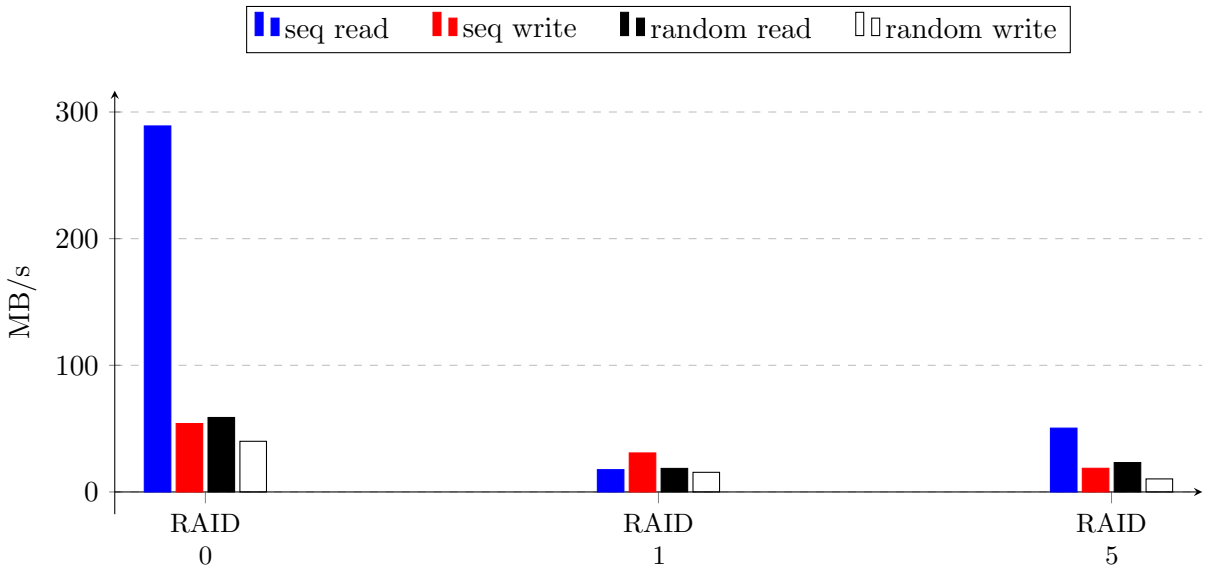

Abbildung 3: BtrFS RAID performance

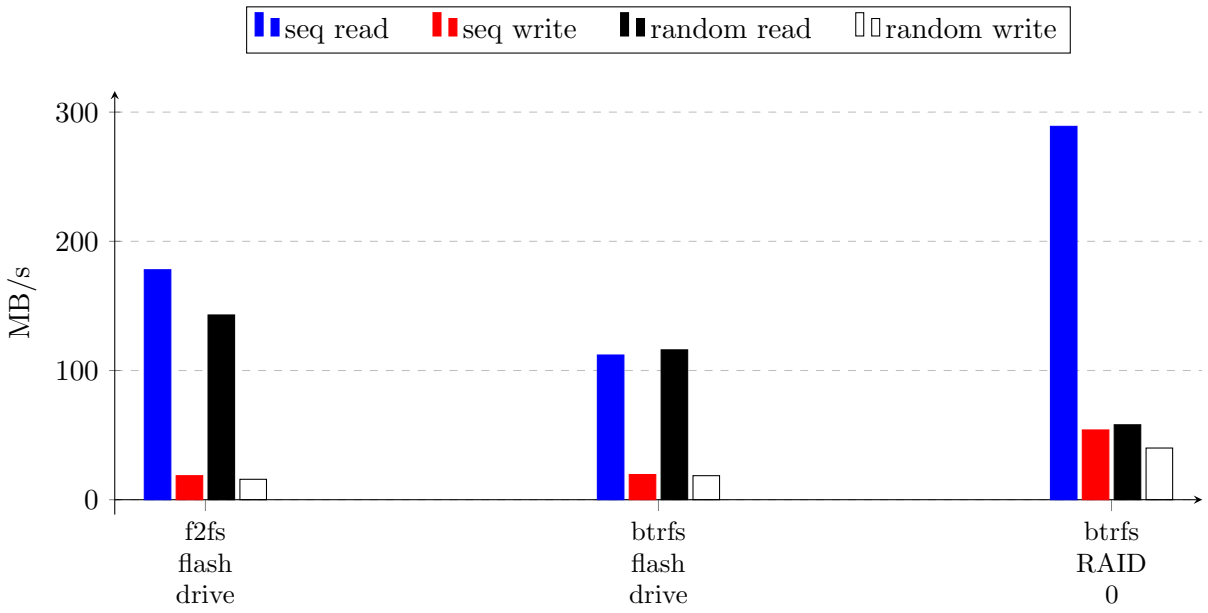

Abbildung 4: F2FS vs. BtrFS vs. BtrFS RAID0