BtrFS & F2FS

Sven Fiergolla & Tobias Dahlem

3. März 2020

Übersicht

Flashspeicher

BtrFS

F2FS

Benchmarks

Besonderheiten des Flashspeichers

- Spezielle Struktur der Speicherzellen

- Adressen Mapping
  - Zuweisung von logischen zu physischen Adressen
  - Häufige Verwendung wegen Charakteristika von Flash-Speichern
- Garbage Collection
  - Alte Daten/Seiten werden als ungültig markiert (Allokierter Speicherplatz)
  - Hoher Aufwand durch kopieren von Blöcken
- Wear Leveling
  - Begrenzte Haltbarkeit von Flash-Zellen (Löschen, Schreiben)
  - Gleichmäßige Abnutzung der Zellen

- Verwendung eines Flash Translation Layer (FTL) im Controller

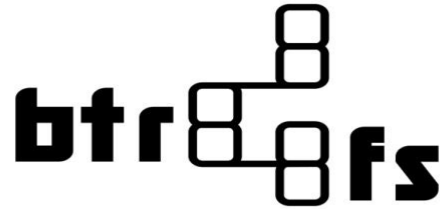
Besonderheiten des Flashspeichers

- Spezielle Struktur der Speicherzellen
- Adressen Mapping
  - Zuweisung von logischen zu physischen Adressen
  - Häufige Verwendung wegen Charakteristika von Flash-Speichern
- Garbage Collection
  - Alte Daten/Seiten werden als ungültig markiert (Allokierter Speicherplatz)
  - Hoher Aufwand durch kopieren von Blöcken
- Wear Leveling
  - Begrenzte Haltbarkeit von Flash-Zellen (Löschen, Schreiben)
  - Gleichmäßige Abnutzung der Zellen
- Verwendung eines Flash Translation Layer (FTL) im Controller

Besonderheiten des Flashspeichers

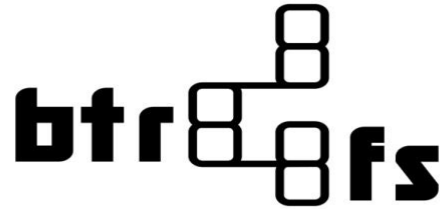
- Spezielle Struktur der Speicherzellen
- Adressen Mapping
  - Zuweisung von logischen zu physischen Adressen
  - Häufige Verwendung wegen Charakteristika von Flash-Speichern
- Garbage Collection
  - Alte Daten/Seiten werden als ungültig markiert (Allokierter Speicherplatz)
  - Hoher Aufwand durch kopieren von Blöcken
- Wear Leveling
  - Begrenzte Haltbarkeit von Flash-Zellen (Löschen, Schreiben)
  - Gleichmäßige Abnutzung der Zellen
- Verwendung eines Flash Translation Layer (FTL) im Controller

BtrFS - Einleitung



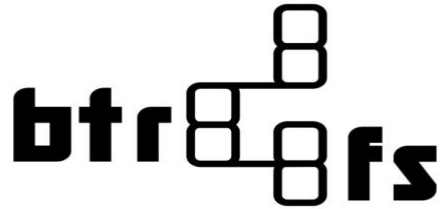
- B-tree file system
  - basiert auf “B-trees, shadowing, and clones” by Ohad Rodeh

BtrFS - Einleitung



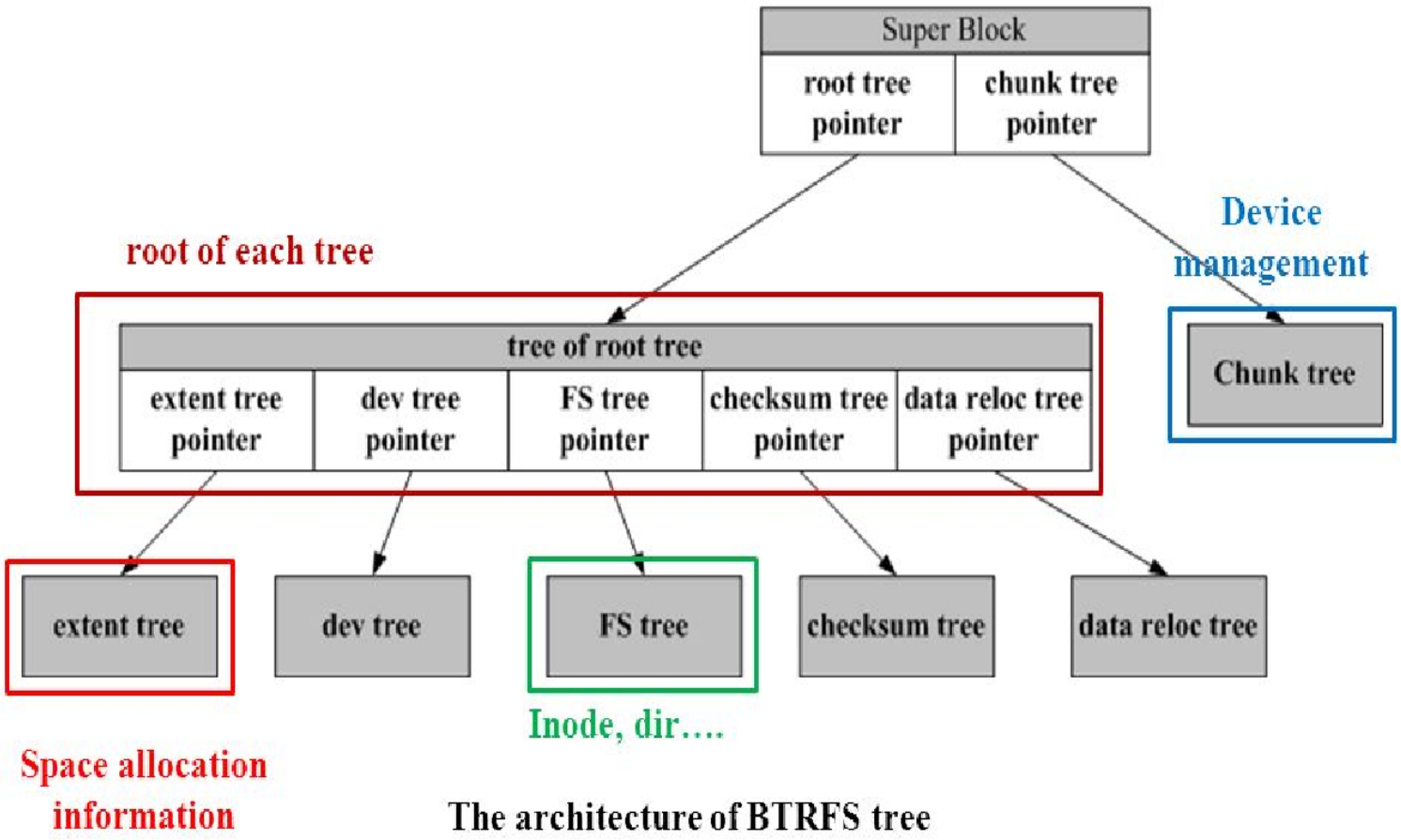
- B-tree file system
  - basiert auf “B-trees, shadowing, and clones” by Ohad Rodeh
- Introduced: Linux kernel 2.6.29, March 2009

BtrFS - Einleitung



- B-tree file system
  - basiert auf “B-trees, shadowing, and clones” by Ohad Rodeh
- Introduced: Linux kernel 2.6.29, March 2009
- Developed by: Facebook, Fujitsu, Fusion-IO, Intel, Linux Foundation, Netgear, Oracle Corporation, Red Hat, STRATO AG, SUSE, ...

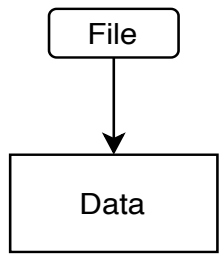
BtrFS - Struktur



The architecture of BTRFS tree

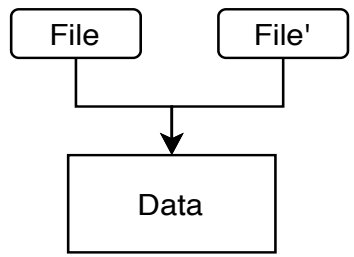
BtrFS - Copy on Write (CoW)

Initiale Datei



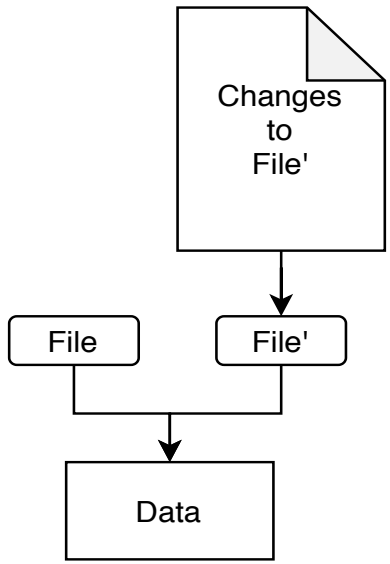
BtrFS - Copy on Write (CoW)

Initiale Datei und unveränderte Kopie

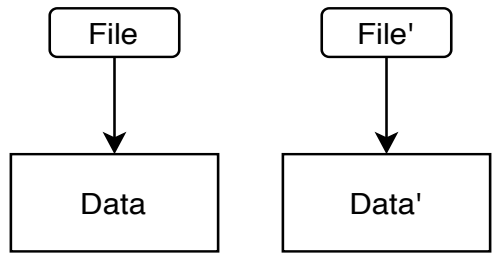


BtrFS - Copy on Write (CoW)

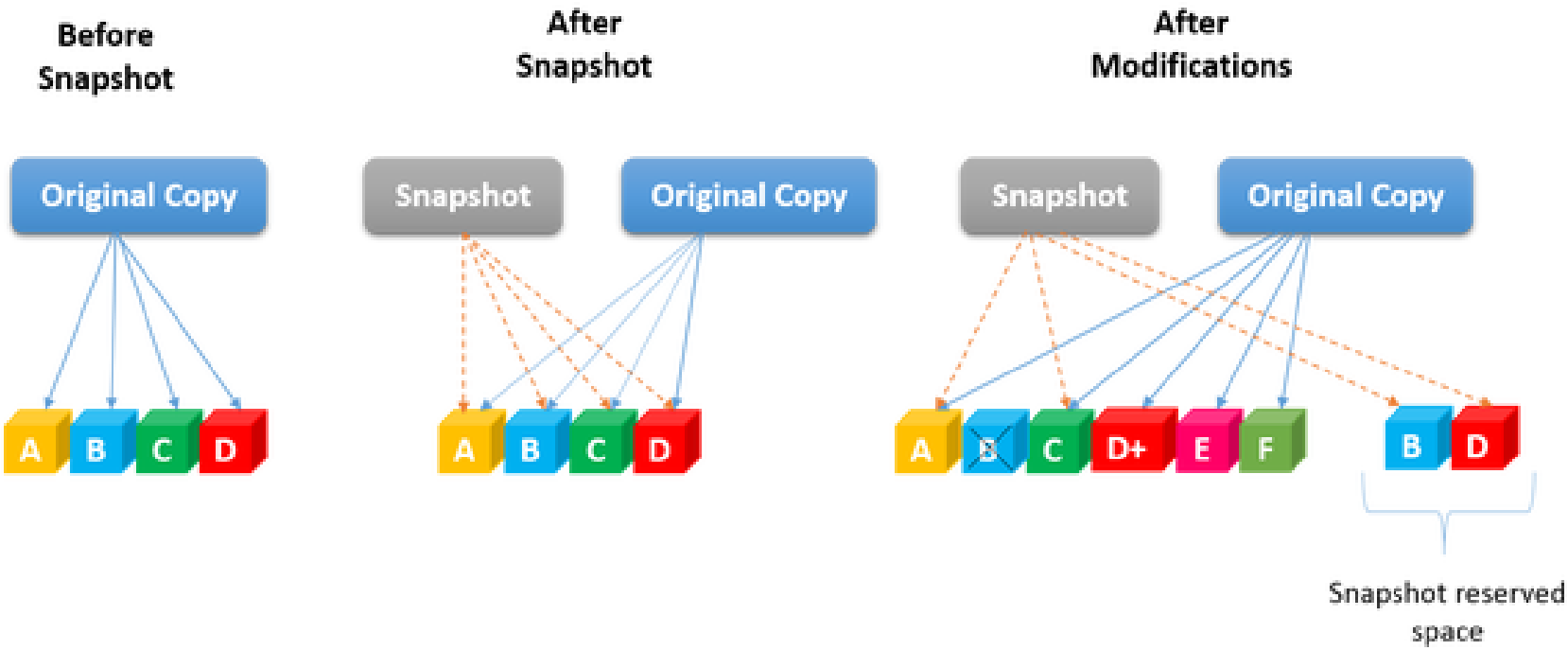
Kopie erfährt Anderungen



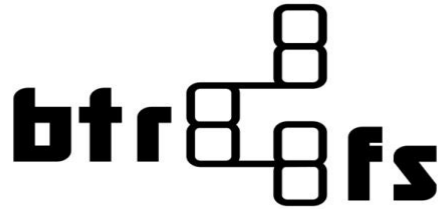
BtrFS - Copy on Write (CoW)



BtrFS - Snapshots



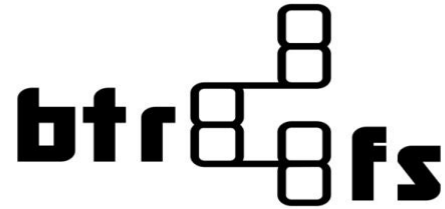
BtrFS - Data & Metadata Checksum



original

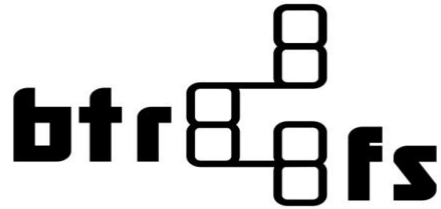
beschädigt

BtrFS - Data & Metadata Checksum



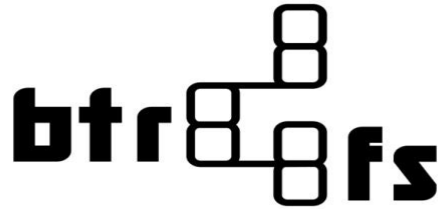
- BtrFS generiert für alle Dateien und Metadaten eine Prüfumme

BtrFS - Data & Metadata Checksum



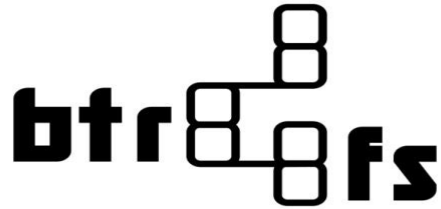
- BtrFS generiert für alle Dateien und Metadaten eine Prüfumme
- Beim lesen eines defekten Extent, wird die Inkonsistenz der Prüfsummen festgestellt

BtrFS - Data & Metadata Checksum



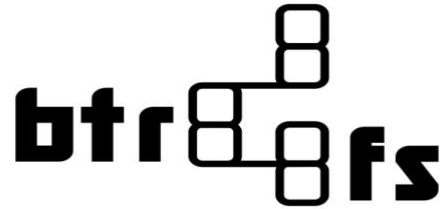
- BtrFS generiert für alle Dateien und Metadaten eine Prüfumme
- Beim lesen eines defekten Extent, wird die Inkonsistenz der Prüfsummen festgestellt
  - mit Mirroring, korrekte Kopie wird gelesen und Defekt wird ersetzt
  - ohne Mirroring, Extent wird verworfen und ein *EIO* zurückgegeben

BtrFS - Data & Metadata Checksum



- BtrFS generiert für alle Dateien und Metadaten eine Prüfumme
- Beim lesen eines defekten Extent, wird die Inkonsistenz der Prüfsummen festgestellt
  - mit Mirroring, korrekte Kopie wird gelesen und Defekt wird ersetzt
  - ohne Mirroring, Extent wird verworfen und ein *EIO* zurückgegeben
- Mit *btrfs scrub* für ganzes Dateisystem möglich

BtrFS - Data & Metadata Checksum

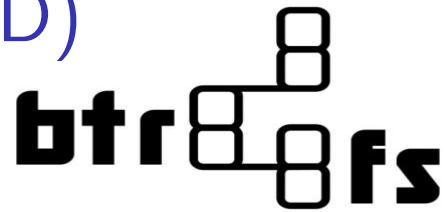


- BtrFS generiert für alle Dateien und Metadaten eine Prüfumme
- Beim lesen eines defekten Extent, wird die Inkonsistenz der Prüfsummen festgestellt
  - mit Mirroring, korrekte Kopie wird gelesen und Defekt wird ersetzt
  - ohne Mirroring, Extent wird verworfen und ein *EIO* zurückgegeben
- Mit *btrfs scrub* für ganzes Dateisystem möglich

dmesg

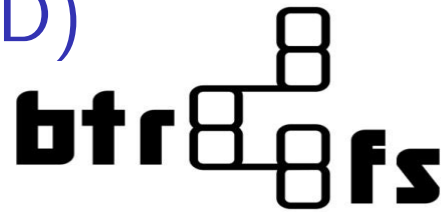
```
[ 1120.891229] verify_parent_transid: 43 callbacks suppressed
[ 1120.891243] parent transid verify failed on 16956989440 wanted 13182 found 12799
[ 1124.851937] parent transid verify failed on 16585043968 wanted 13145 found 12357
[ 1124.885429] btrfs read error corrected: ino 1 off 16977842176 (dev /dev/sdd sector 2921768)
```

Redundant Array of Independent Disks (RAID)



- Ermöglicht Organisation mehrerer Datenträger zu einem logischen Volume
- Erlaubt effizientes redundantes Speichern
  - Rekonstruktion verlorener Daten bei Ausfall eines Dateiträgers

Redundant Array of Independent Disks (RAID)

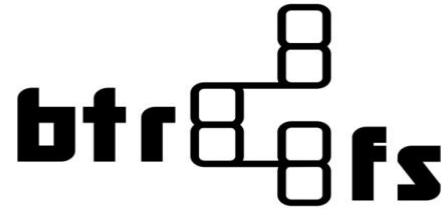


- Ermöglicht Organisation mehrerer Datenträger zu einem logischen Volume
- Erlaubt effizientes redundantes Speichern
  - Rekonstruktion verlorener Daten bei Ausfall eines Dateiträgers

Nachteile:

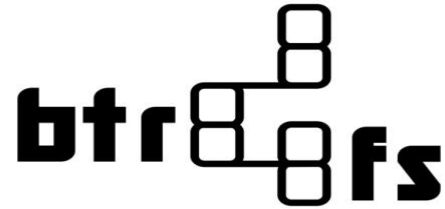
- Fixe Plattengröße
- Modifizierung erfordert Restriping

BtrFS - RAID



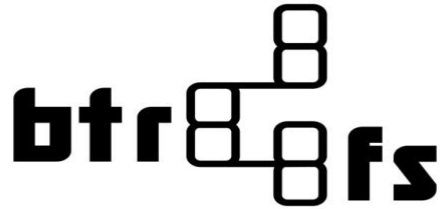
- RAID 0 (keine Parität) & 1,5 (einfache Parität) & 6 (mehrfache Parität)

BtrFS - RAID



- RAID 0 (keine Parität) & 1,5 (einfache Parität) & 6 (mehrfache Parität)
- Keine einheitliche Plattengröße

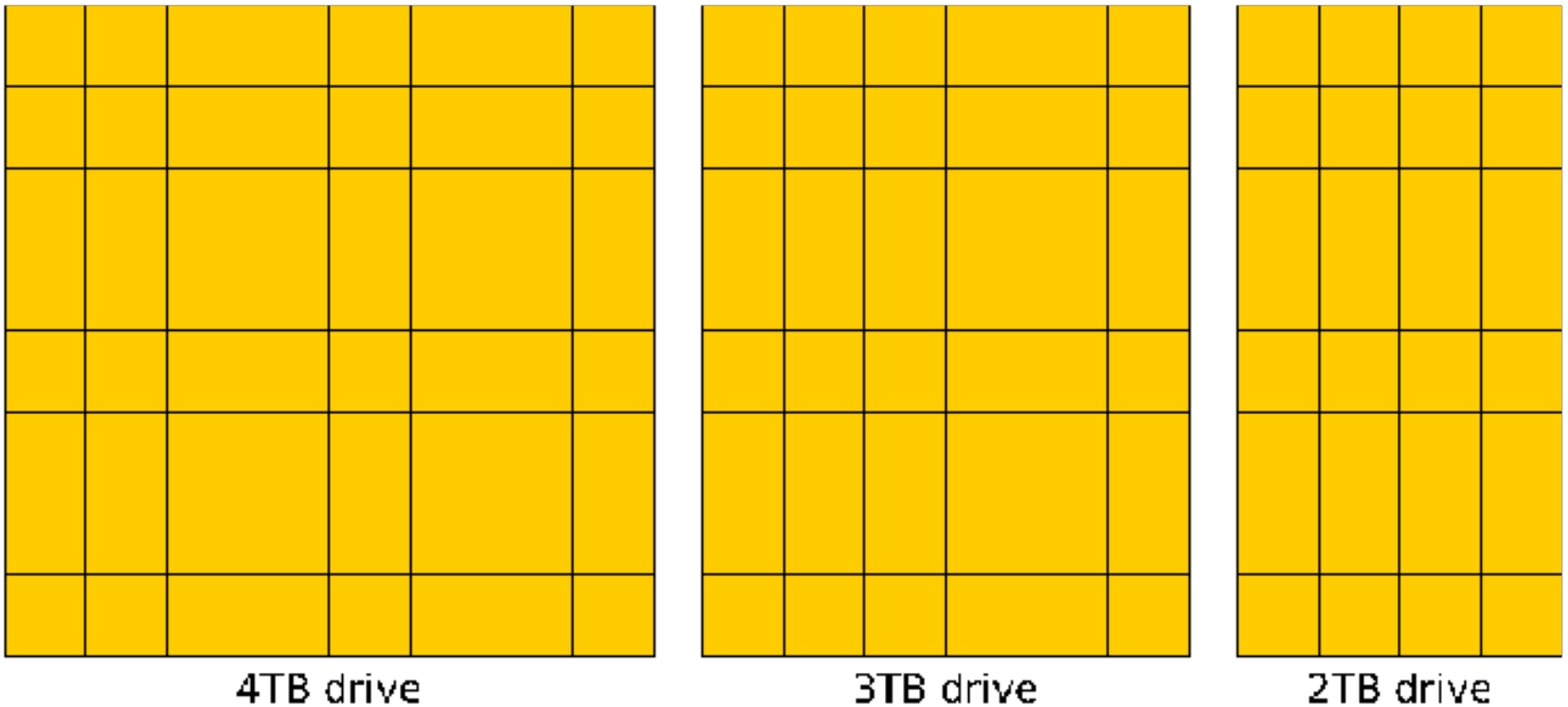
btrfs - RAID



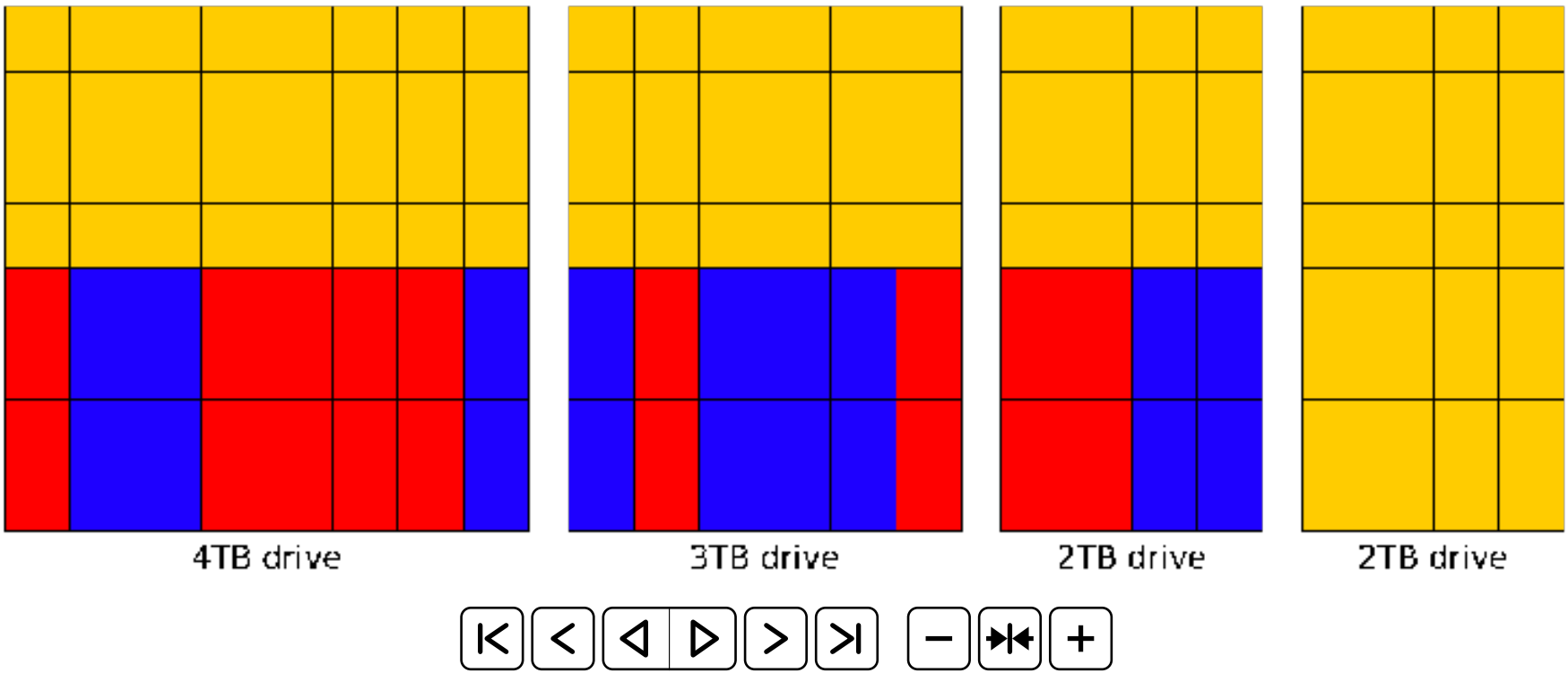
- RAID 0 (keine Parität) & 1,5 (einfache Parität) & 6 (mehrfache Parität)
- Keine einheitliche Plattengröße
- Dynamisches erweitern

```
mkfs.btrfs -d <mode> -m <mode> <dev1> <dev2> ...
mount /dev/<disk x> /<mount point>
btrfs device add /dev/<disk y> /<mount point>
```

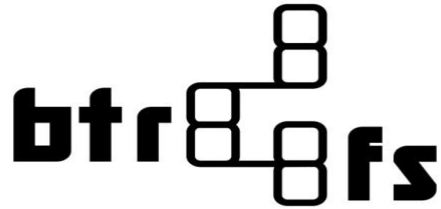
BtrFS - RAID



BtrFS - RAID

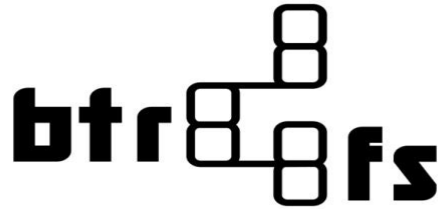


BtrFS - Weitere Features



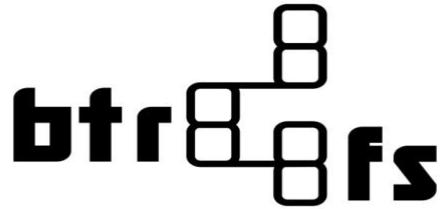
- Snapshots, read-only Snapshots
- btrfs RAID (raid0, raid1, raid10, raid5 & raid6)

BtrFS - Weitere Features



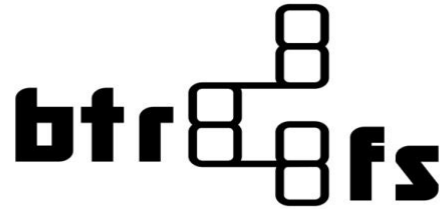
- Snapshots, read-only Snapshots
- btrfs RAID (raid0, raid1, raid10, raid5 & raid6)
- Komprimierung auf Filesystem Ebene
  - verfügbare Algorithmen: *zlib, lzo & zstd*
  - *mount -o compress /dev/sdx*
  - *btrfs filesystem defrag -r /path*

BtrFS - Weitere Features



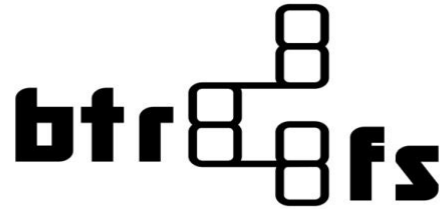
- Snapshots, read-only Snapshots
- btrfs RAID (raid0, raid1, raid10, raid5 & raid6)
- Komprimierung auf Filesystem Ebene
  - verfügbare Algorithmen: *zlib, lzo & zstd*
  - *mount -o compress /dev/sdx*
  - *btrfs filesystem defrag -r /path*
- Subvolumes
  - eigene interne Dateisystem roots
  - *btrfs subvolume create name*

BtrFS - Nachteile



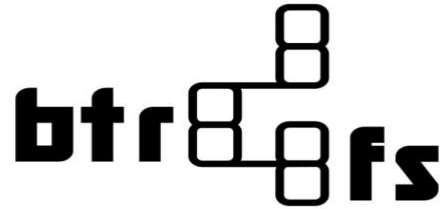
- B^+ Baum kann unbalanciert werden
  - manuelles rebalancieren

BtrFS - Nachteile



- B^+ Baum kann unbalanciert werden
  - manuelles rebalancieren
- CoW Mechanismus führt zu Inkonsistenzen OLTP Workload, fragwürdig für Datenbanken

BtrFS - Nachteile



- B^+ Baum kann unbalanciert werden
  - manuelles rebalancieren
- CoW Mechanismus führt zu Inkonsistenzen OLTP Workload, fragwürdig für Datenbanken
- Berechnung des benötigten Speicherplatzes mit Snapshots
  - btrfs filesystem df / anstatt df /

F2FS

- Flash-Dateisystem von Samsung (veröffentlicht 2012)
- Entwickelt nur für Flash-Speicher (SD-Karte, SSDs, eMMC-Karten)
- Ziel: Optimierung der Performance und Lebenszeit von Flash-Speichern
- Entwickelt als Open-Source Projekt

- Verfolgt den Ansatz eine Log-structured File System (append-only logging)
- Arbeitet nicht auf ,,raw" Flash-Zellen (Benötigt einen FTL)
- Viele Möglichkeiten zur Anpassung des Systems
- Verwendung von iNodes und Datenblöcken (Ähnlich zu UNIX)

- Verfügbar ab Linux Kernel 3.8
- Verwendung in Huawei (2016), Galaxy Note 10, Google Nexus

F2FS

- Flash-Dateisystem von Samsung (veröffentlicht 2012)
- Entwickelt nur für Flash-Speicher (SD-Karte, SSDs, eMMC-Karten)
- Ziel: Optimierung der Performance und Lebenszeit von Flash-Speichern
- Entwickelt als Open-Source Projekt

- Verfolgt den Ansatz eine Log-structured File System (append-only logging)
- Arbeitet nicht auf ,,raw" Flash-Zellen (Benötigt einen FTL)
- Viele Möglichkeiten zur Anpassung des Systems
- Verwendung von iNodes und Datenblöcken (Ähnlich zu UNIX)

- Verfügbar ab Linux Kernel 3.8
- Verwendung in Huawei (2016), Galaxy Note 10, Google Nexus

F2FS

- Flash-Dateisystem von Samsung (veröffentlicht 2012)
- Entwickelt nur für Flash-Speicher (SD-Karte, SSDs, eMMC-Karten)
- Ziel: Optimierung der Performance und Lebenszeit von Flash-Speichern
- Entwickelt als Open-Source Projekt

- Verfolgt den Ansatz eine Log-structured File System (append-only logging)
- Arbeitet nicht auf ,,raw" Flash-Zellen (Benötigt einen FTL)
- Viele Möglichkeiten zur Anpassung des Systems
- Verwendung von iNodes und Datenblöcken (Ähnlich zu UNIX)

- Verfügbar ab Linux Kernel 3.8
- Verwendung in Huawei (2016), Galaxy Note 10, Google Nexus

F2FS - Flash-friendly on-disk Layout

- Einheiten: Blöcke, Segmente, Sektionen, Zonen
- Orientierung an FTL-Einheiten um Kosten zu Vermeiden
- Metadaten:
  - Random Writes: Vorhalten in Arbeitsspeicher (Bei Checkpoints schreiben)
- Haupt-Speicherbereich:
  - Aufgeteilt in Standardmäßig 4KB Blocks (Jeder Block ist Node- oder Data-Block)
  - Node- und Data-Blocks liegen in verschiedenen Segmenten

F2FS - Flash-friendly on-disk Layout

- Einheiten: Blöcke, Segmente, Sektionen, Zonen
- Orientierung an FTL-Einheiten um Kosten zu Vermeiden
- Metadaten:
  - Random Writes: Vorhalten in Arbeitsspeicher (Bei Checkpoints schreiben)
- Haupt-Speicherbereich:
  - Aufgeteilt in Standardmäßig 4KB Blocks (Jeder Block ist Node- oder Data-Block)
  - Node- und Data-Blocks liegen in verschiedenen Segmenten

Figure: on-disk Layout F2FS

F2FS - Flash-friendly on-disk Layout

- Einheiten: Blöcke, Segmente, Sektionen, Zonen
- Orientierung an FTL-Einheiten um Kosten zu Vermeiden
- Metadaten:
  - Random Writes: Vorhalten in Arbeitsspeicher (Bei Checkpoints schreiben)
- Haupt-Speicherbereich:
  - Aufgeteilt in Standardmäßig 4KB Blocks (Jeder Block ist Node- oder Data-Block)
  - Node- und Data-Blocks liegen in verschiedenen Segmenten

Figure: on-disk Layout F2FS

F2FS - Besonderheiten I

- Multi-Head Logging
  - Mehrere aktive Logsegmente parallel (Standard 6)
  - Parallele Verwendung durch Architektur möglich (multi-Streaming Interface)
  - Unterscheidung der Daten in hot/warm/cold Schema (Update Frequenz)

- Kosten-Effiziente Index Struktur
  - Verwendung einer neuartigen Indexes: note adress table (NAT)
  - Zur Vermeidung des ,,wandering tree" Problems
    - Nur Update des direct Node Block und NAT
    - Reduktion der Updates um Schreiboperationen zu sparen

F2FS - Besonderheiten I

- Multi-Head Logging
  - Mehrere aktive Logsegmente parallel (Standard 6)
  - Parallele Verwendung durch Architektur möglich (multi-Streaming Interface)
  - Unterscheidung der Daten in hot/warm/cold Schema (Update Frequenz)

- Kosten-Effiziente Index Struktur
  - Verwendung einer neuartigen Indexes: note adress table (NAT)
  - Zur Vermeidung des ,,wandering tree" Problems
    - Nur Update des direct Node Block und NAT
    - Reduktion der Updates um Schreiboperationen zu sparen

F2FS - Besonderheiten II

- Adaptive logging
  - Append-only Logging : Standardmäßig (random writes werden sequentiell)
  - Threaded Logging : Verwendung bei hoher Auslastung (random writes)

- Garbage Collection
  - On-Demand: Wenn nicht genügend Speicherplatz verfügbar ist
    - Greedy: Auswahl des Opfersegments mit wenigsten gültigen Blöcken
  - Background: Bei geringer Auslastung des Systems von Kernel ausgeführt
    - Kosten-Effizient: Auswahl durch Segment-Alter und Anzahl gültiger Blöcke

F2FS - Besonderheiten II

- Adaptive logging
  - Append-only Logging : Standardmäßig (random writes werden sequentiell)
  - Threaded Logging : Verwendung bei hoher Auslastung (random writes)

- Garbage Collection
  - On-Demand: Wenn nicht genügend Speicherplatz verfügbar ist
    - Greedy: Auswahl des Opfersegments mit wenigsten gültigen Blöcken
  - Background: Bei geringer Auslastung des Systems von Kernel ausgeführt
    - Kosten-Effizient: Auswahl durch Segment-Alter und Anzahl gültiger Blöcke

F2FS - Bewertung

- Vorteile
  - Optimierung der Zusammenarbeit von FTL und Dateisystem
  - Vermeidung des Wandering Tree Problems
  - Anpassung des Dateisystems an System-Status
  - Hohe Anzahl an Parametern um Dateisystem anzupassen
- Nachteile
  - Nur für Flash-Speicher (mit einem FTL)
  - FTL Qualität wichtiges Kriterium
  - Initialer hoher belegter Speicherplatz durch Metadaten
  - Hohe CPU-Belastung beim Schreiben von Dateien

F2FS - Bewertung

- Vorteile
  - Optimierung der Zusammenarbeit von FTL und Dateisystem
  - Vermeidung des Wandering Tree Problems
  - Anpassung des Dateisystems an System-Status
  - Hohe Anzahl an Parametern um Dateisystem anzupassen
- Nachteile
  - Nur für Flash-Speicher (mit einem FTL)
  - FTL Qualität wichtiges Kriterium
  - Initialer hoher belegter Speicherplatz durch Metadaten
  - Hohe CPU-Belastung beim Schreiben von Dateien

Benchmarks - F2FS Results

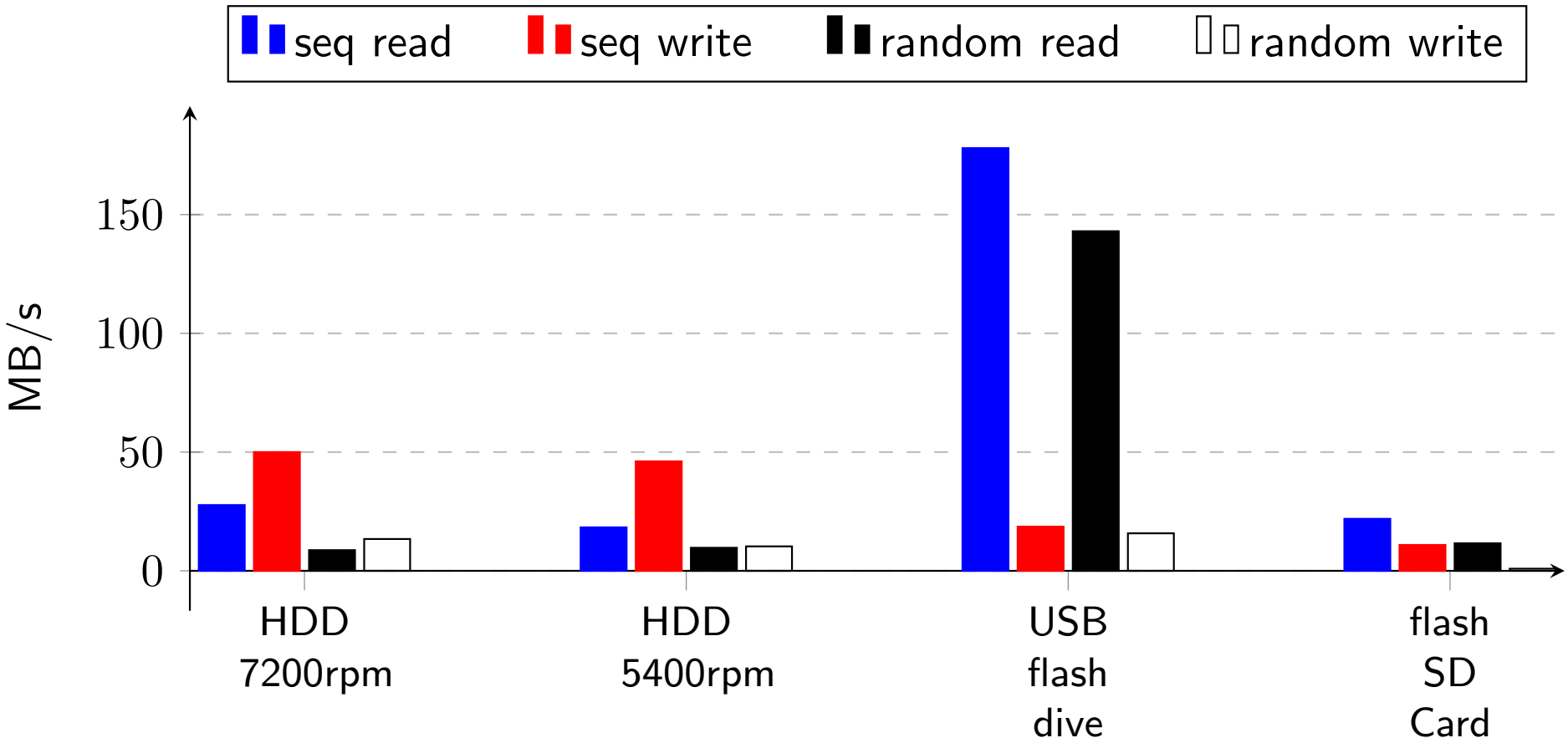


Figure: F2FS raw performance

Benchmarks - BtrFS Results

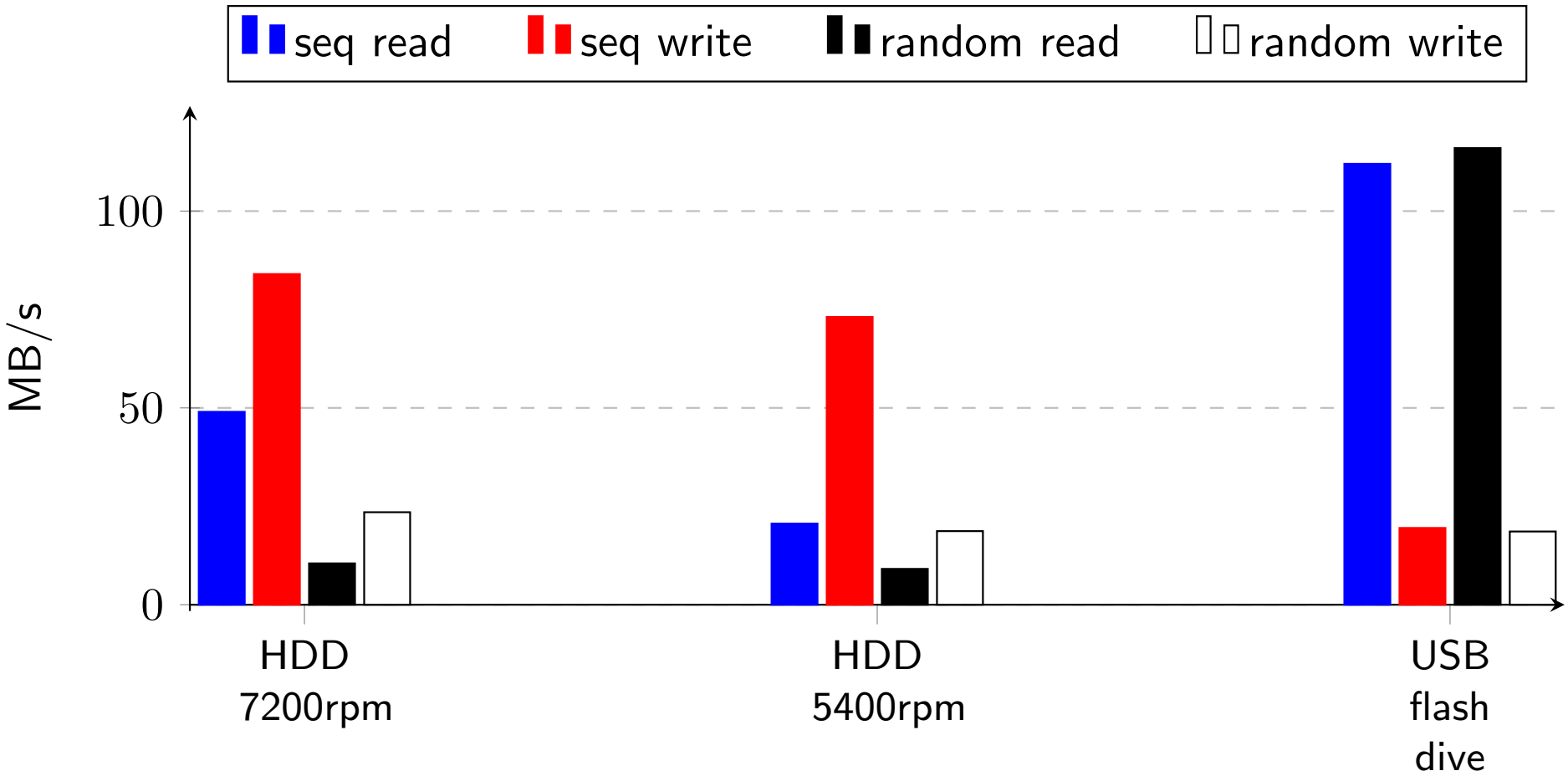


Figure: BtrFS raw performance

Benchmarks - Btrfs Results

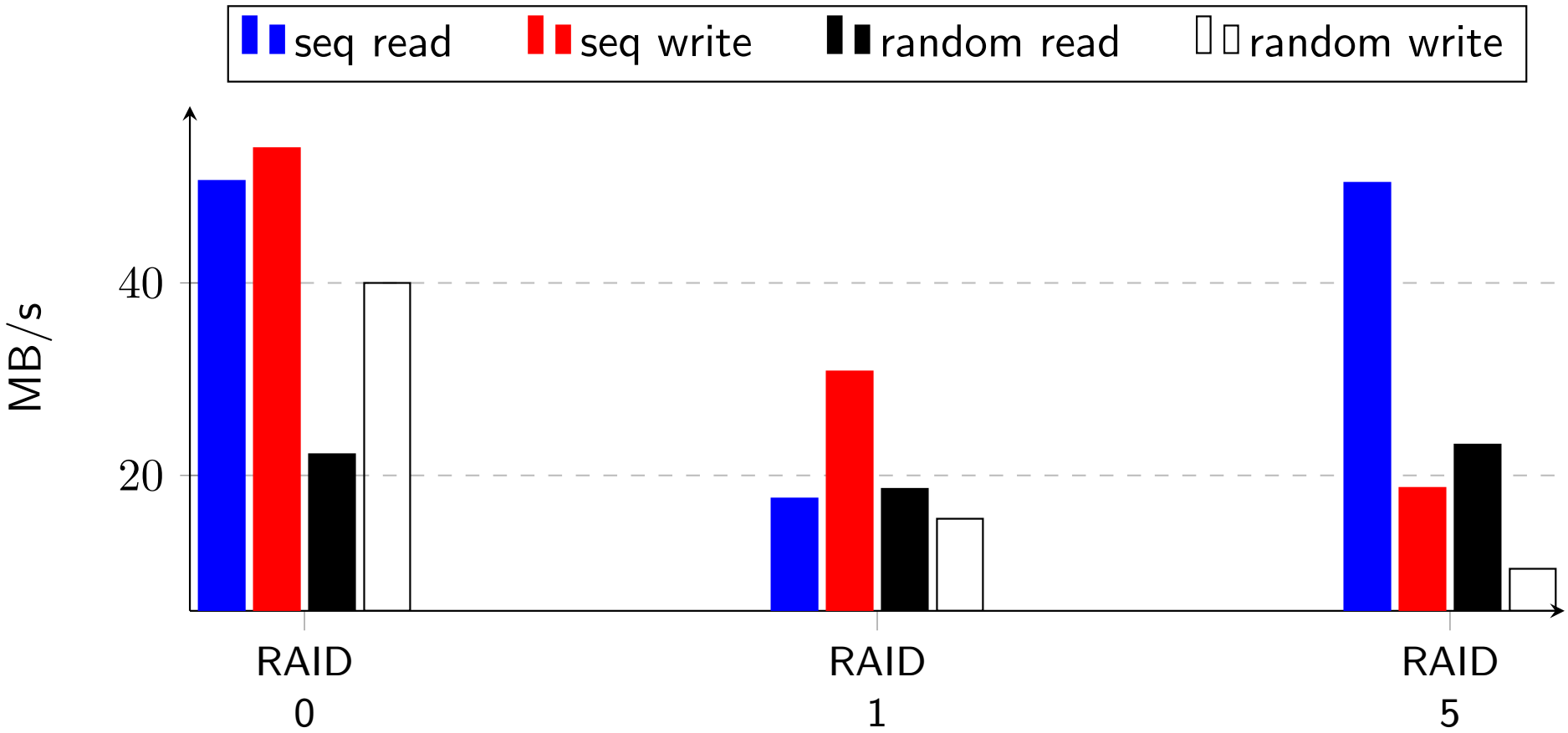


Figure: BtrFS RAID performance

Benchmarks - BtrFS vs. F2Fs

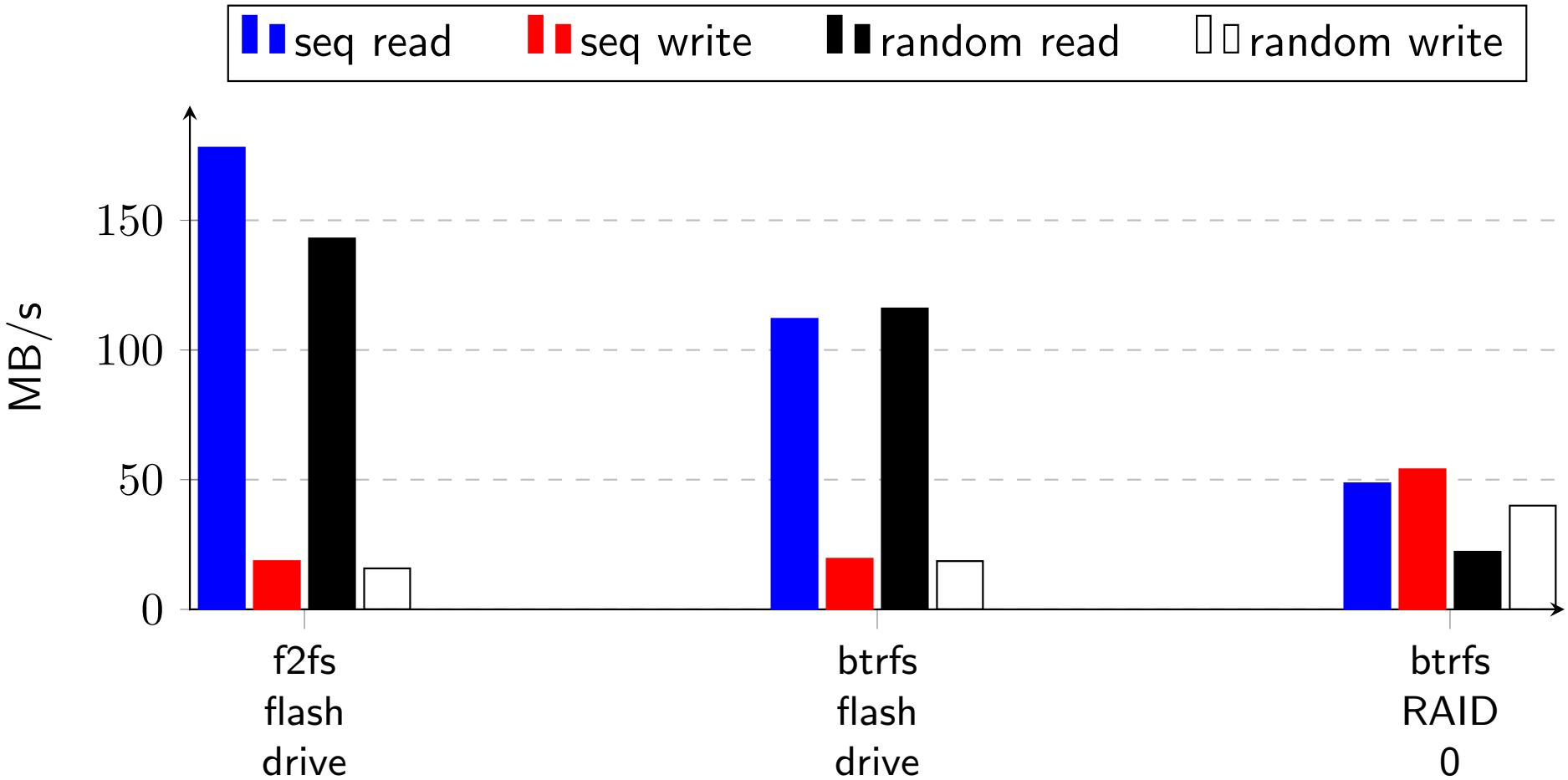


Figure: F2FS vs. BtrFS vs. BtrFS RAID0

Literatur

F2FS archLinux. https://wiki.archlinux.org/index.php/F2FS. Accessed: 2020-02-10.

F2FS Wikipedia (English). https://en.wikipedia.org/wiki/F2FS. Accessed: 2020-02-10.

Flash Wikipedia. https://en.wikipedia.org/wiki/Flash_memory. Accessed: 2020-02-10.

Changman Lee et al. "F2FS: A new file system for flash storage". In: *13th USENIX Conference on File and Storage Technologies (FAST 15)*. 2015, pp. 273–286.

Ohad Rodeh. "B-trees, shadowing, and clones". In: *ACM Transactions on Storage (TOS)* 3.4 (2008), pp. 1–27.