Enabling Random Access in Universal Compressors

Seminar "Supercomputer Forschung und Innovation"

Felix Pusch

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

2023-01-17

Gliederung (Agenda)

Motivation

Warum Komprimieren?

- Speicher ist nicht unendlich
- Schnellerer Transfer
- Bessere Auslastung von Ressourcen

Komplementäre Strategien zur Optimierung

- **Deduplikation**: System speichert identische Dateien nur einmal (ZFS, Git)
- **Kompression**: Eliminierung von identischen Daten in einer Datei

Kompression

Universal Data Compressors

- generisch, ohne Vorabwissen über die Datei
- LZ-Schema, Deflate, zstd, brotli

Eigenschaften

- Verlustfrei (*lossless*)
- Kompressionsfaktor
- Kompressionsgeschwindigkeit
- Dekompressionsgeschwindigkeit

Kompression

Zugriffsarten

- Lesen der gesamten Datei (Sequenziell)
- Lesen eines (zufälligen) Segments der gesamten Datei (Random Access)

Fundamentales Problem

- gesamte Datei muss dekomprimiert werden um einen Teil zu lesen
- gerade bei großen Dateien impraktikabel (obwohl genau diese sinnvoll zu komprimieren sind)

Paper

Enabling Random Access in Universal Compressors [VZL21]

- Rasmus Vestergaard, Qi Zhang, and Daniel E. Lucani
- Aarhus University, Dänemark
- ICCN 2021: IEEE International Workshop on Intelligent Cloud Computing and Networking

Andere Veröffentlichungen zu

- Generalized Deduplication [VZL19a]
- Lossless Compression of Time Series Data [VZL19b]
- Randomly Accessible Compression for Time Series Data [VLZ20]

Ziel

Entwicklung eines Random Access Compressors der

1. kleine Resultate erzeugt, also *gut* komprimiert
2. eine hohe Kompressions- & Dekompressionsgeschwindigkeit hat, also *schnell* ist
3. eine hohe Zugriffsgeschwindigkeit für kleine (*zufällig gewählte*) Segmente aus dem Archiv hat

Priorität zwischen Ziel (1) und (2) werden je nach Compressor unterschiedlich gelöst

Ziel

Was bedeutet

3. eine hohe Zugriffsgeschwindigkeit für kleine (*zufällige*) Segmente aus dem Archiv hat

Zugriff auf einen Teil der Datei

- unabhängig von der Gesamtgröße der Datei (und des Archivs)
- abhängig von der Größe des angefragten Segments, sprich r bytes in $O(r)$ Zeit

Konzept

a Teile Datei in *gleich große Segmente*

b Komprimiere einzelne Segmente mit einem Universal Data Compressor

c Serialisiere in *zwei Dateien*

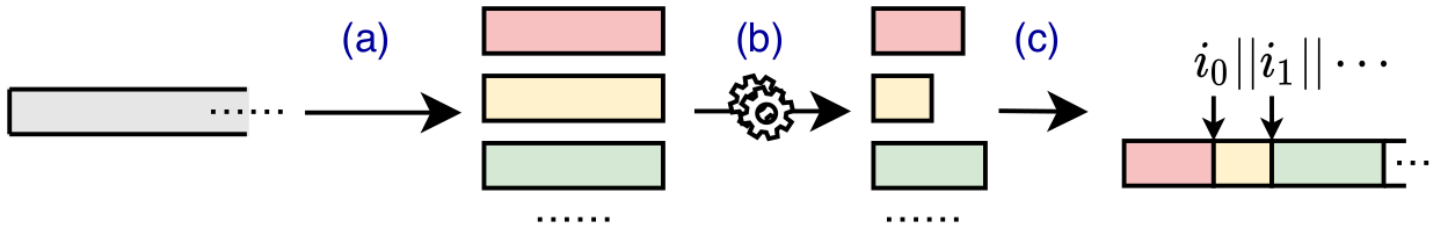

Abbildung: Concept [VZL21]

Konzept

Zwei Dateien?

- Komprimiertes Archiv (mit gleicher Reihenfolge der Segmente)
- Indices der Grenzen der Segmente, da nicht gleich groß

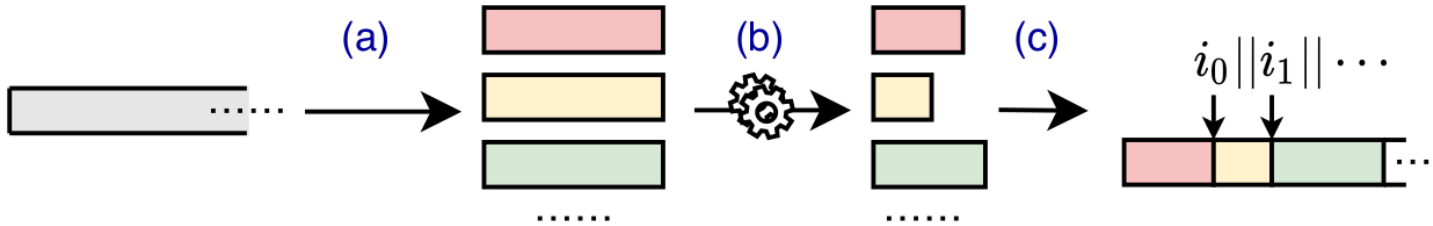

Abbildung: Concept [VZL21]

Konzept

Dekompression

1. Lese Grenze der Segmente aus Indexfile
2. Lese komprimiertes Segment aus Archiv
3. Dekomprimiere mit Universal Data Compressor

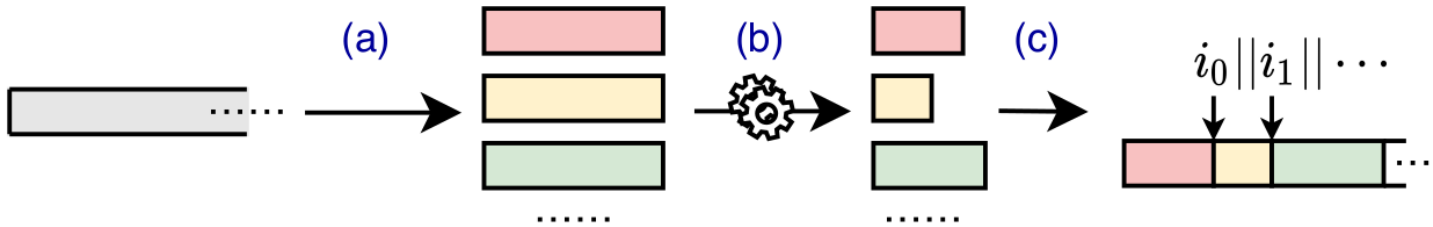

Abbildung: Concept [VZL21]

Konsequenzen des Konzepts

1. Segmente, da kleiner, lassen sich schlechter komprimieren als Gesamtdatei (Kompressionsfaktor)
2. Zusätzlicher Berechnungsschritt (Geschwindigkeit)
3. Granularität des Random Access ist auf Segmentgröße beschränkt
   $=>$ ein Segment muss komplett dekomprimiert werden um *zufällig ein Byte* zu lesen

Ziel

- Zeigen, dass obige Effekte in der Praxis nicht problematisch sind

Hardware

Hardware: 3.7 GHz Intel Xeon, 32 GB RAM, Mechanische HDD

Tabelle: Datensatz, adaptiert von [VZL21]

Name	Datatype	File Size
BGL	**Blue Gene/L supercomputer log**	**743.2 MB**
DNA	A sequenced genome	173.4 MB
Audio	Human speech recording	319.5 MB

Algorithmen

Tabelle: Kompressionsalgorithmen, adaptiert von [VZL21]

Name	Fokus	Autor
zlib	Generisch	IETF RFC1950
LZFSE	Generisch	Apple
zstd	Generisch	Facebook
Snappy	Geschwindigkeit	Google
LZ4	Geschwindigkeit	Yann Collet
LZ4HC	Komprimierung	Yann Collet
Brotli	Komprimierung	Google
bzip2	Komprimierung	Julian Seward

Kompressionsfaktor

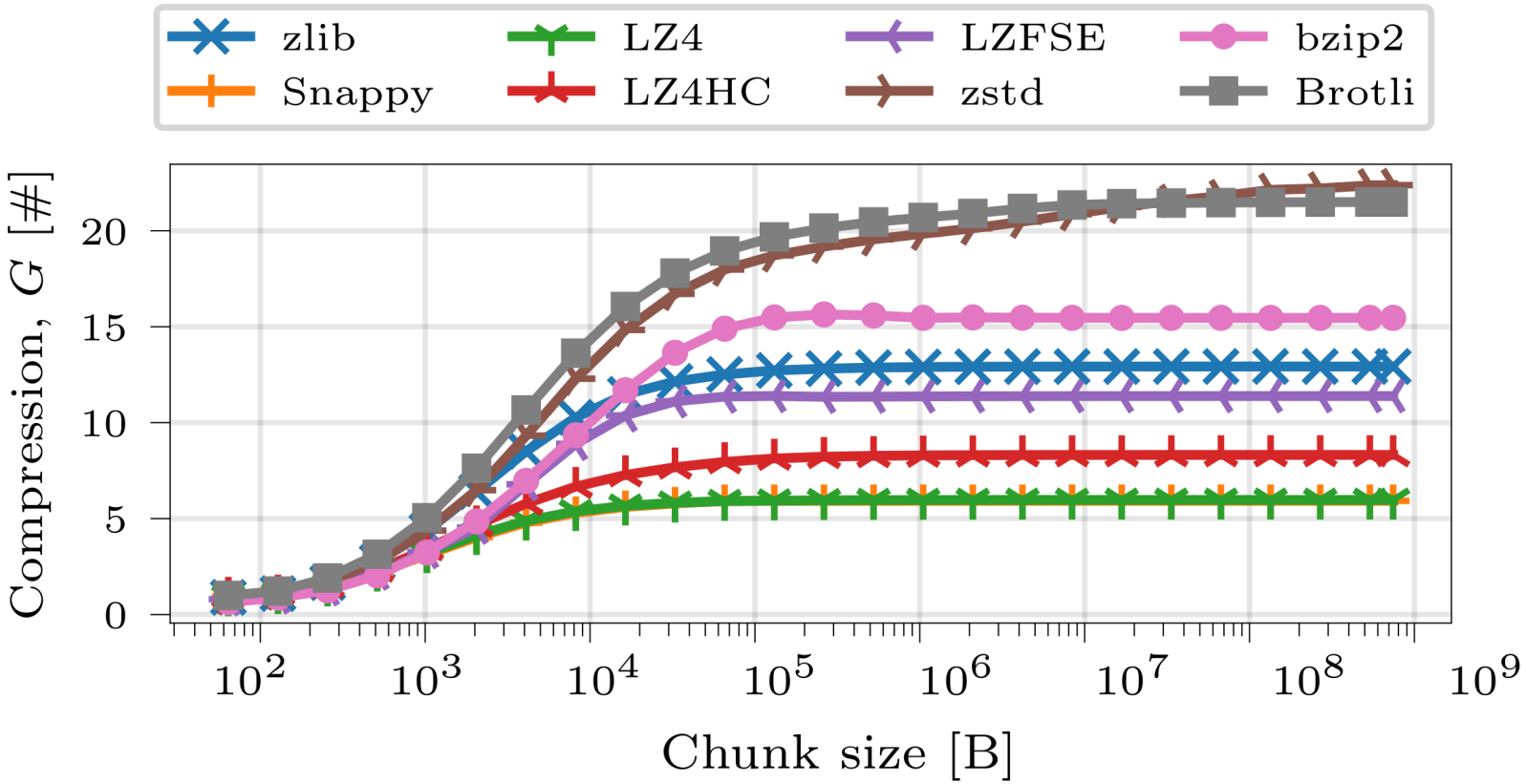

Abbildung: Compression gain, adaptiert von [VZL21]

Kompressionsfaktor

1. Chunks, da kleiner, lassen sich schlechter komprimieren als Gesamtdatei
   $\Longrightarrow$ ab ca. 100 kB Segmentgröße kaum problematisch

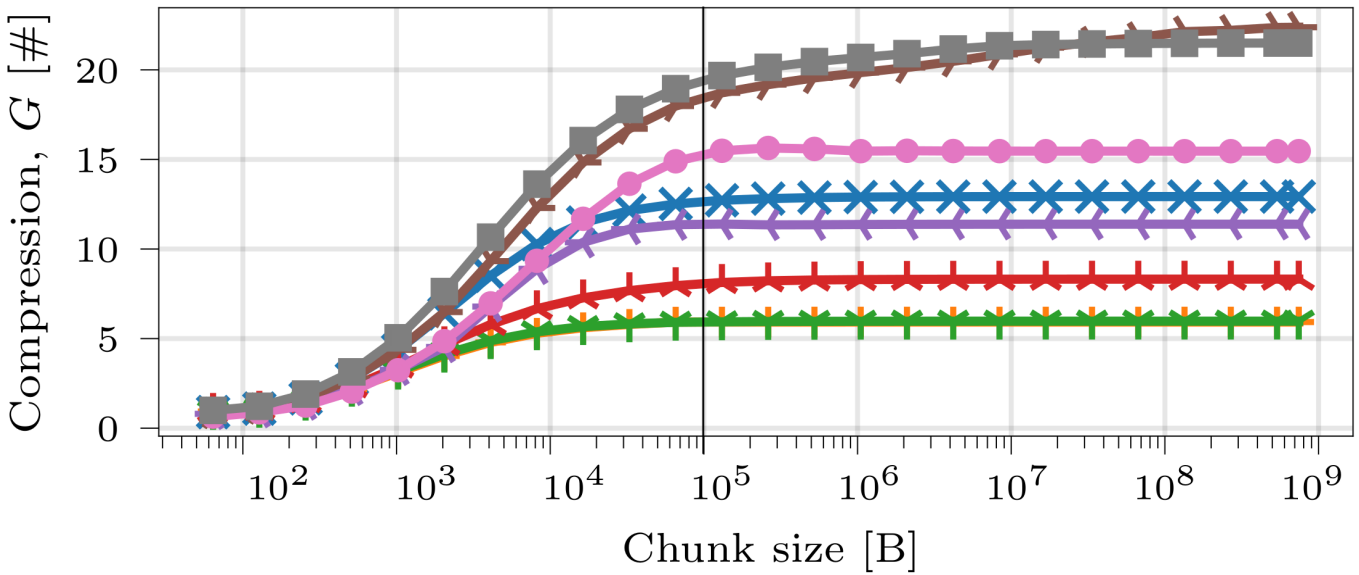

Abbildung: Compression gain, adaptiert von [VZL21]

Kompressionsgeschwindigkeit

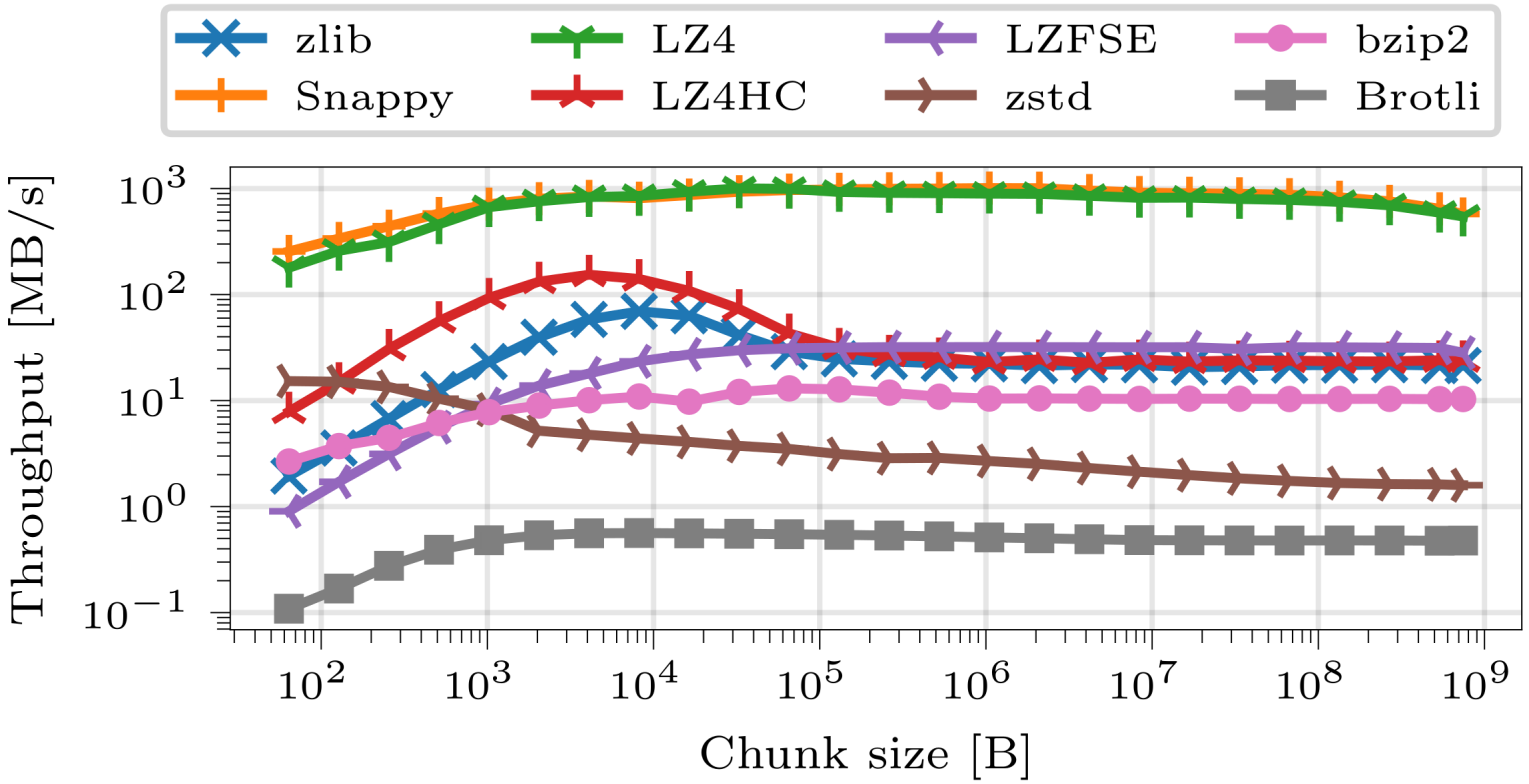

Abbildung: Kompressionsgeschwindigkeit, adaptiert von [VZL21]

Kompressionsgeschwindigkeit

2 Zusätzlicher Berechnungsschritt (Overhead)
$\Rightarrow$ ab ca. 100 kB Segmentgröße kaum problematisch

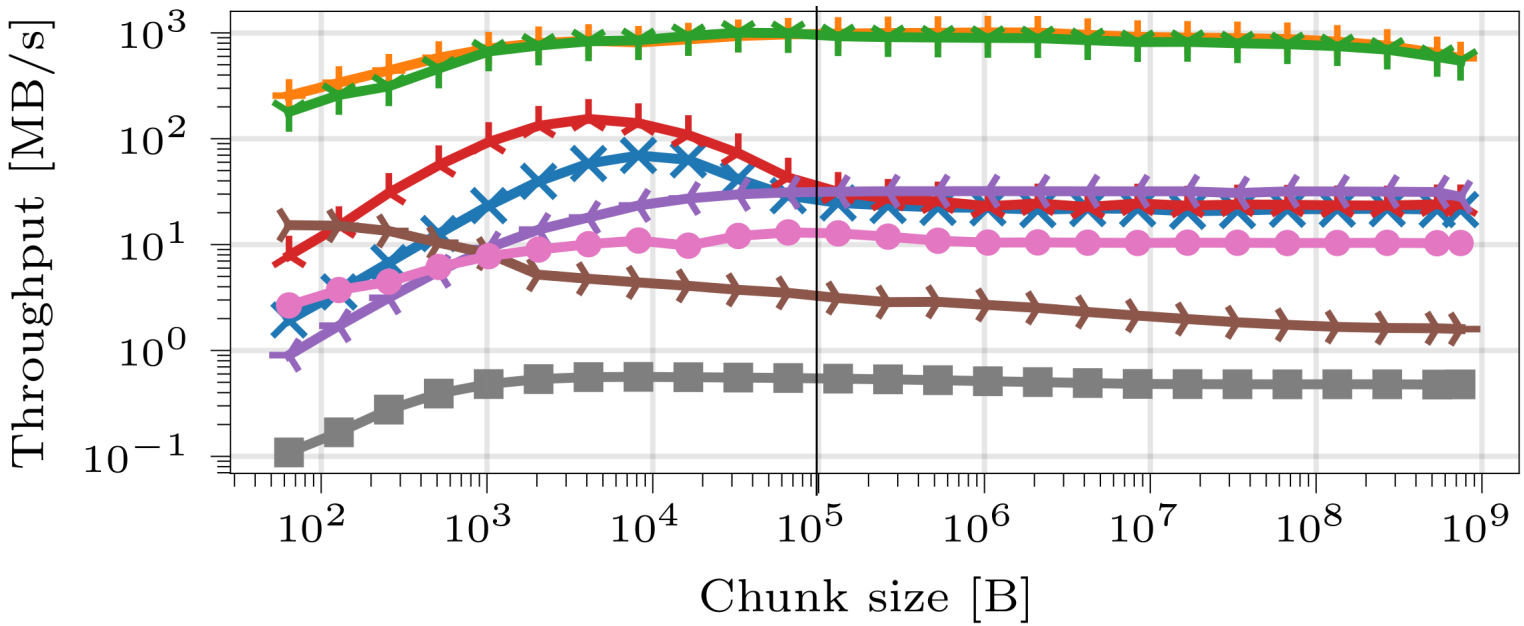

Abbildung: Kompressionsgeschwindigkeit, adaptiert von [VZL21]

Dekompressionsgeschwindigkeit

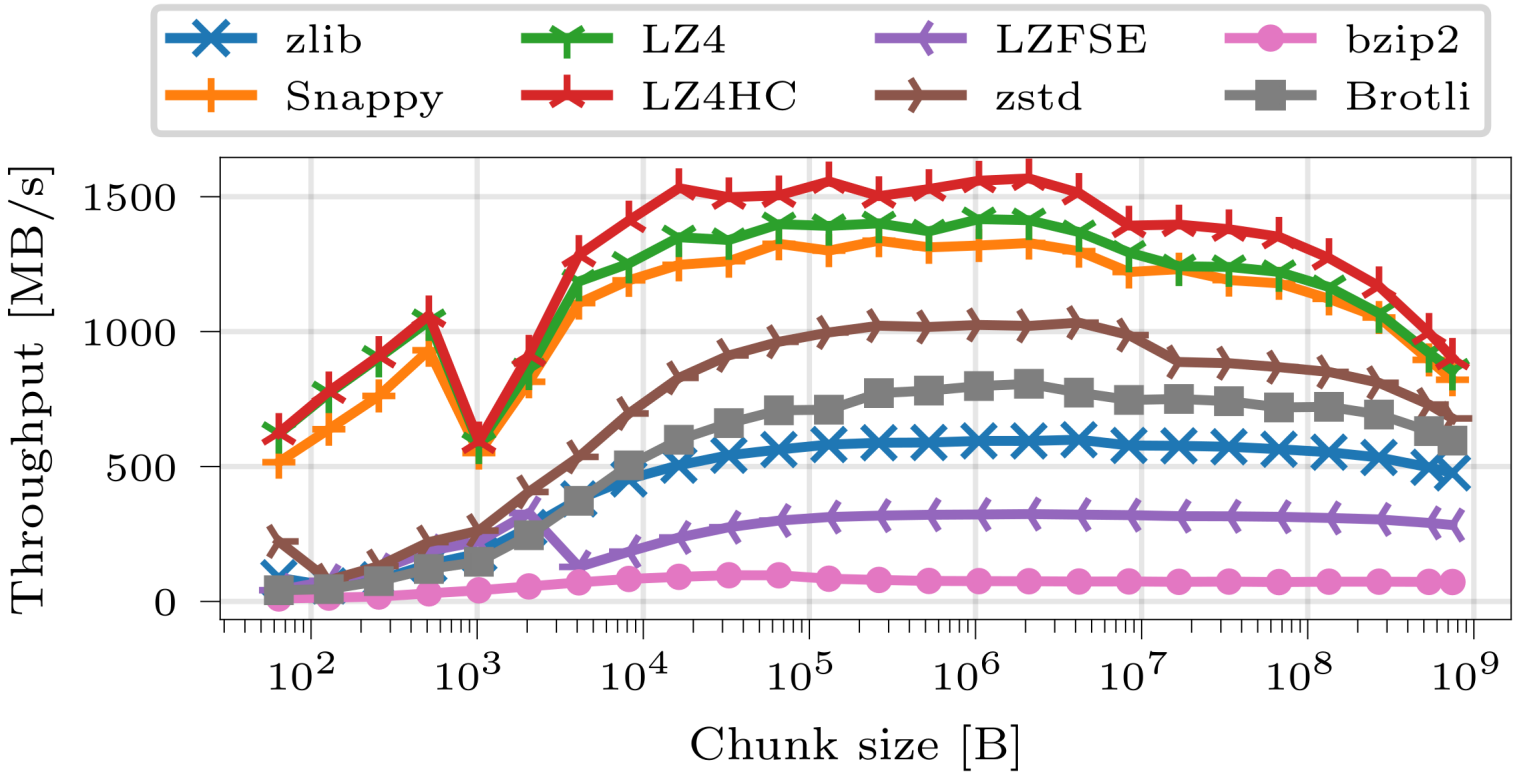

Abbildung: Dekompressionsgeschwindigkeit, adaptiert von [VZL21]

Dekompressionsgeschwindigkeit

2 Zusätzlicher Berechnungsschritt (Overhead)
$\Rightarrow$ ab ca. 100 kB Segmentgröße kaum problematisch

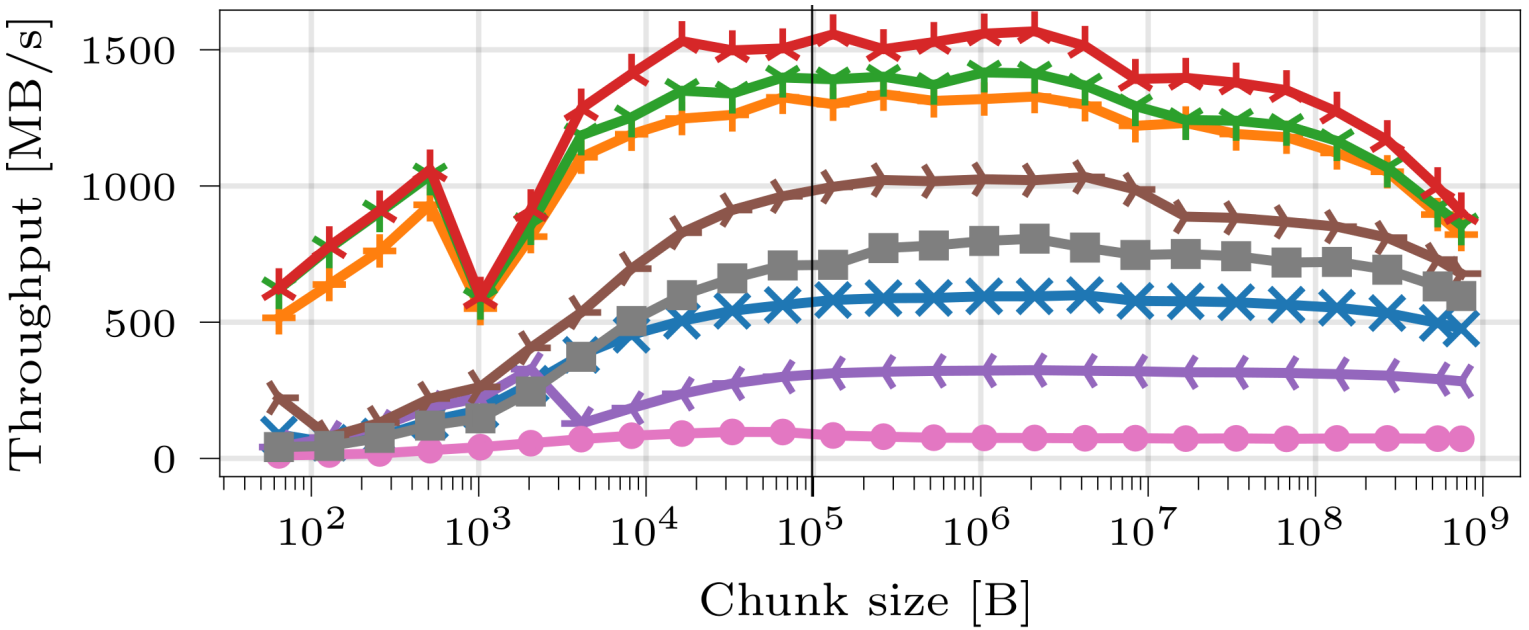

Abbildung: Dekompressionsgeschwindigkeit, adaptiert von [VZL21]

Random Access

- 1000 r-byte Anfragen, variable Segmentgröße
- Beginn an zufälligem Startpunkt in der Quelldatei
- Nicht an komprimierten Segmentgrenzen ausgerichtet
- Daten im Cache (warm) oder von Festplatte (kalt)
- Verglichen mit unkomprimiertem Zugriff

Nicht Komprimiert - Kalter Cache

- ≈ konstante Abfragen/s bis zur Pagegröße

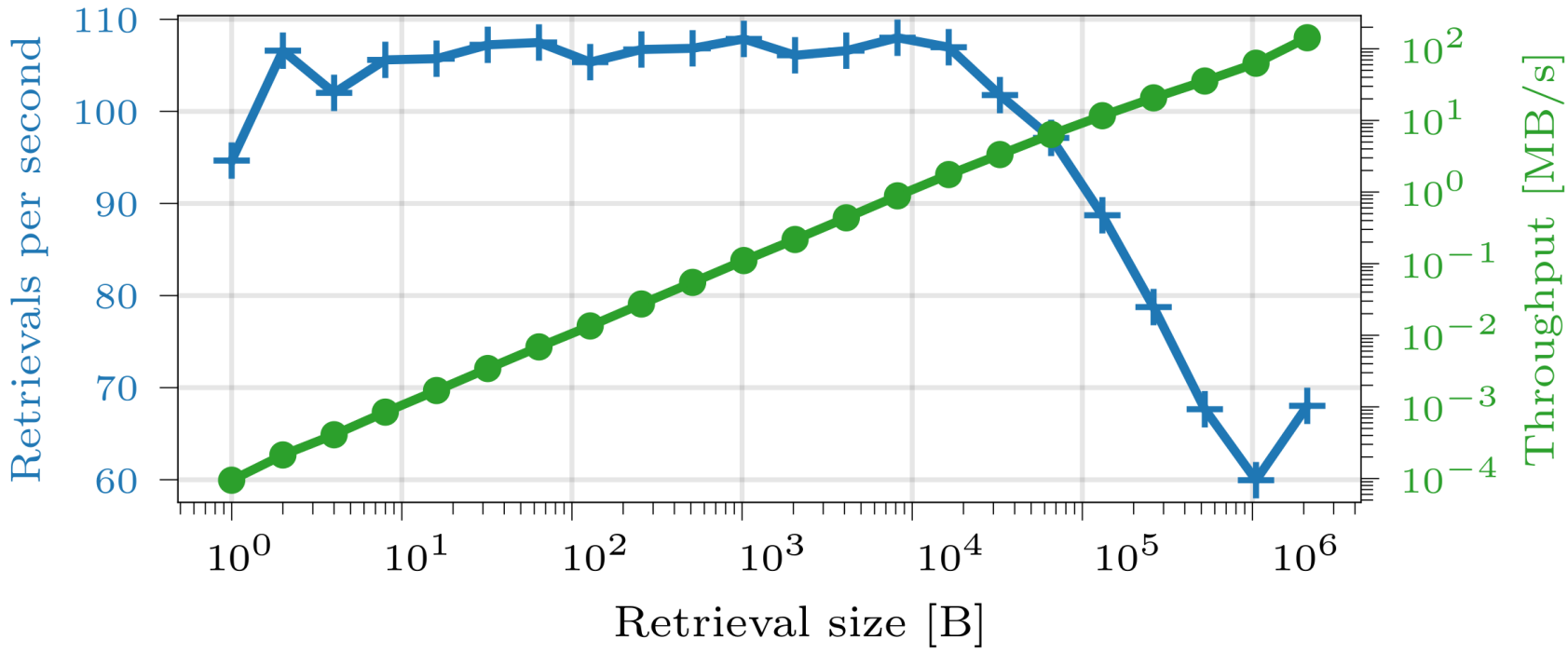

Abbildung: Uncompressed Random Access, Cold Cache, adaptiert von [VZL21]

Nicht Komprimiert - Warmer Cache

- deutlich schneller als aus dem Cache

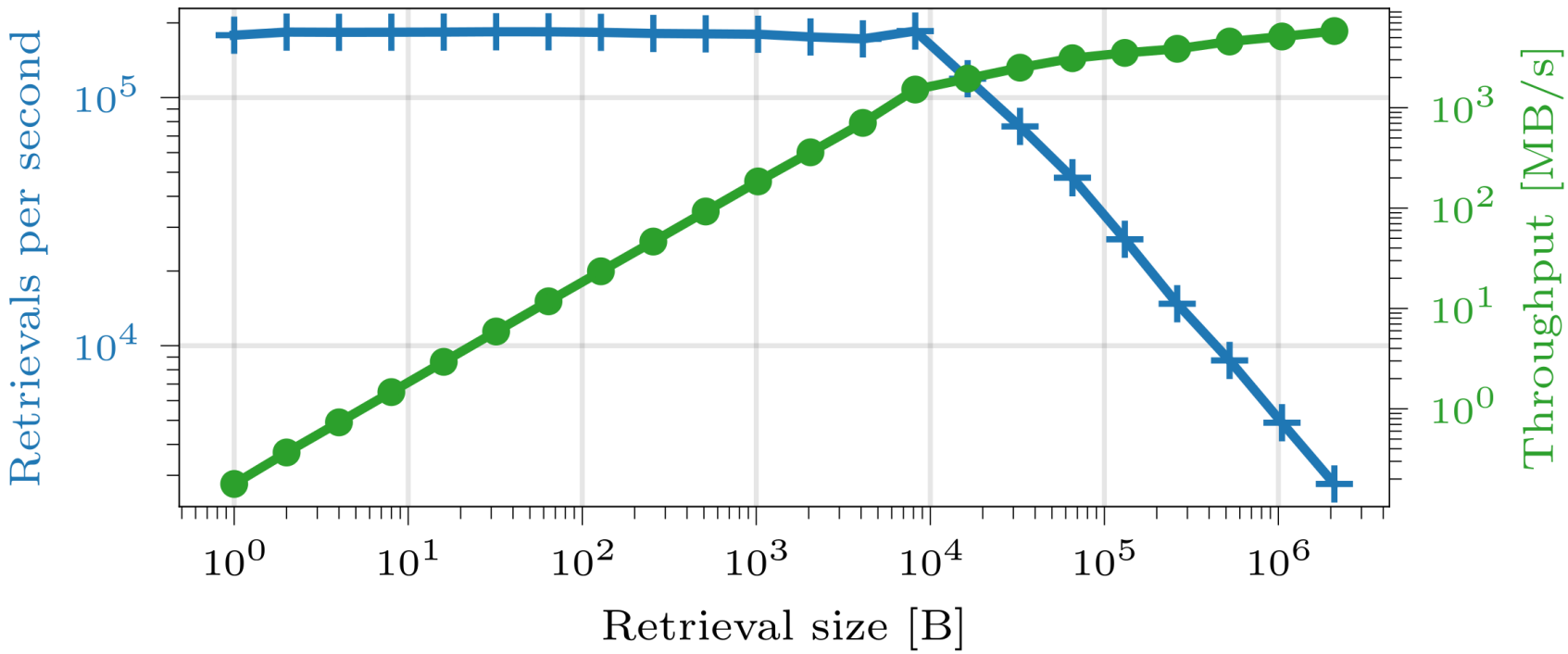

Abbildung: Uncompressed Random Access, Warm Cache, adaptiert von [VZL21]

Komprimiert - Variable Segmentgröße

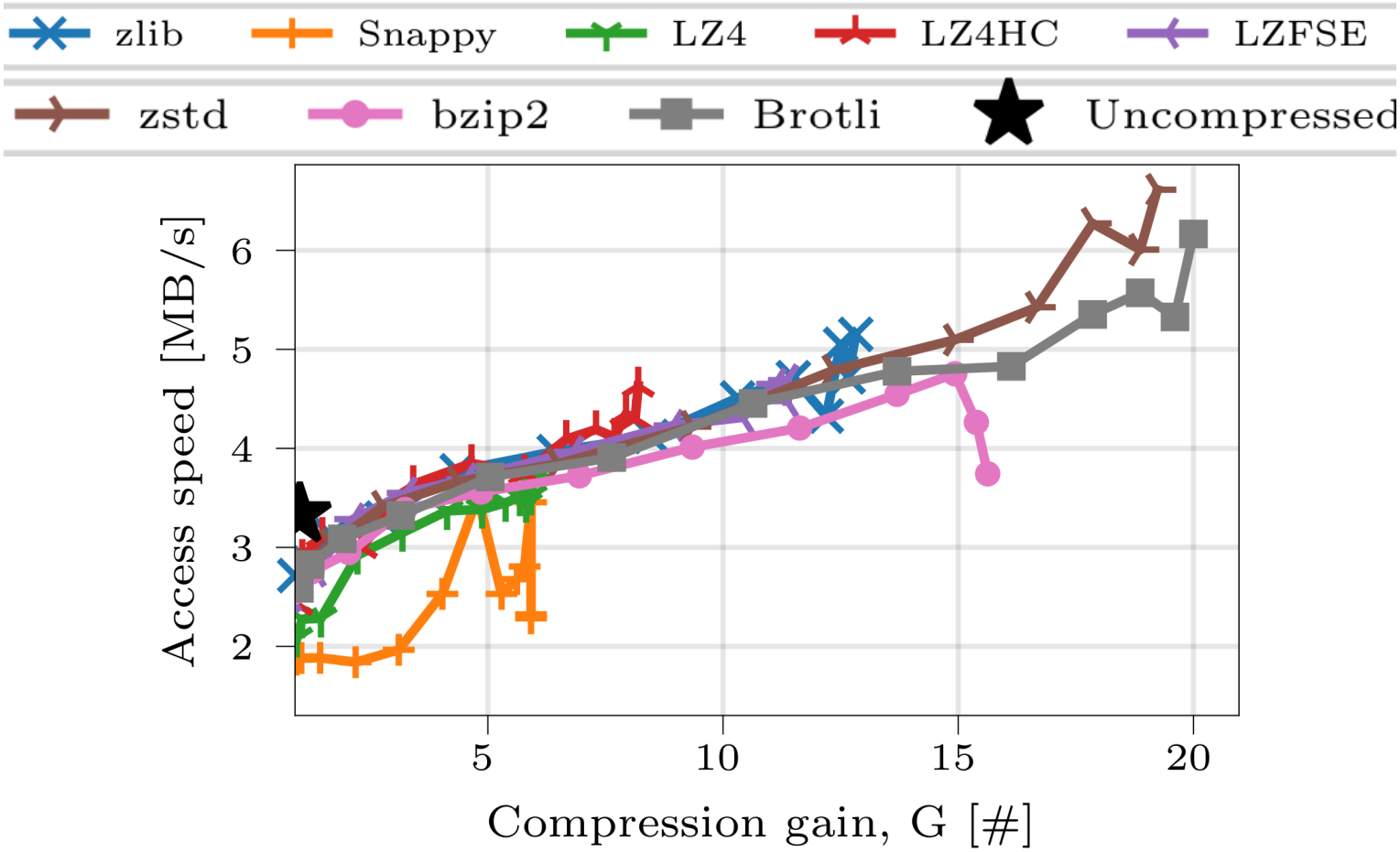

Abbildung: Cold Cache, 32KB Data Random Access Speed, Variable Chunk Size adaptiert von [VZL21]

Komprimiert - Variable Segmentgröße

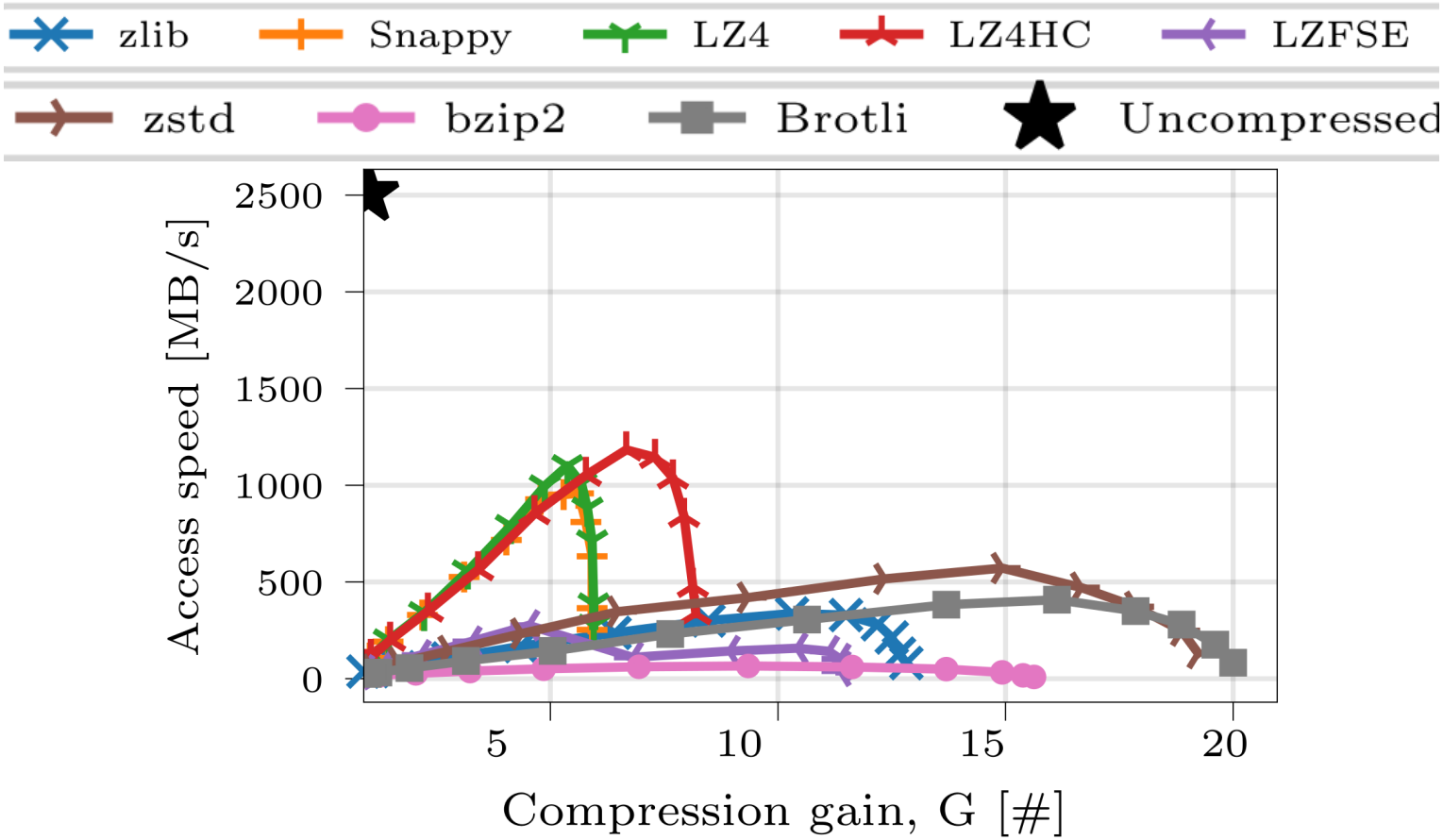

Abbildung: Warm Cache, 32KB Data Random Access Speed, Variable Chunk Size adaptiert von [VZL21]

Komprimiert - Variable Segmentgröße

- kalter Cache: komprimierter Zugriff schneller als nicht komprimierter
  $=>$ Empfehlung: Wahl eines stark komprimierenden Algorithmus
- warmer Cache: nicht komprimierter Zugriff deutlich schneller als komprimierter
  $=>$ Empfehlung: Wahl eines schnellen Algorithmus
- Generell: LZ4HC und zstd guter Kompromiss

Zusammenfassung

- Generischer Ansatz um zufälligen Zugriff auf komprimierte Daten zu ermöglichen ohne gesamte Datei zu dekomprimieren
- Ansatz zeigt kaum Verlust an Kompressionsfaktor oder Geschwindigkeit bei adäquater Segmentgröße
- Je nach Einsatz: zufälliger Zugriff auf komprimierte Daten schneller, oder langsamer, als auf Rohdaten

Meinung

- Paper sehr gut, verständlich geschrieben
- Testmethodik gut erklärt
- Keine Tests bzw. Bewertung von Systemen mit „schnellem“ Speicher (Solid State): evtl. ähnlich zu Cache

Meinung

- Test in HPC Umfeld mit *schnellem* Storage und damit verbunden auch mehr Variablen
- Interessant im Einsatz für verteilte, parallele Dateisystem $=>$ zufälliger Zugriff müsste bei idealer Segmentierung nur zu einem Ziel

Literatur I

[VLZ20] Rasmus Vestergaard, Daniel E. Lucani, and Qi Zhang. A randomly accessible lossless compression scheme for time-series data. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 2145–2154, 2020.

[VZL19a] Rasmus Vestergaard, Qi Zhang, and Daniel E. Lucani. Generalized deduplication: Bounds, convergence, and asymptotic properties. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2019.

Literatur II

[VZL19b] Rasmus Vestergaard, Qi Zhang, and Daniel E. Lucani. Lossless compression of time series data with generalized deduplication. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2019.

[VZL21] Rasmus Vestergaard, Qi Zhang, and Daniel E. Lucani. Enabling random access in universal compressors. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2021.