Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Efficient Metadata Indexing for HPC Storage Systems

Maximilian Keiff

14. Dezember 2021

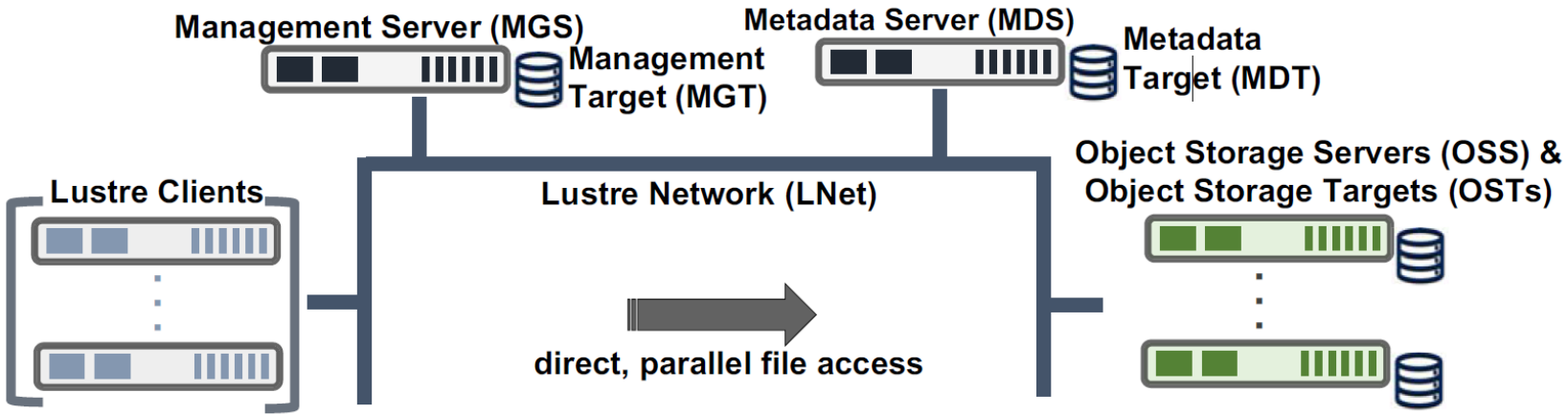
Seminar Supercomputer: Forschung und Innovation

Agenda

- Motivation
  - Was sind Metadaten?
  - Wie funktioniert Metadaten-Indexing?
- Brindexer
  - Partitionierung
  - Indexer
  - Re-Indexer
  - Query Interface
- Auswertung
  - Vergleich der System Calls
  - Performance und Systemauslastung

Was sind Metadaten?

- Informationen zu Dateien (Größe, Besitzer, Rechte, Zugriffzeiten, Links, Layout, …)
- Gespeichert in Inode Files
- Mindestens 10x so viele Attribute wie Dateien



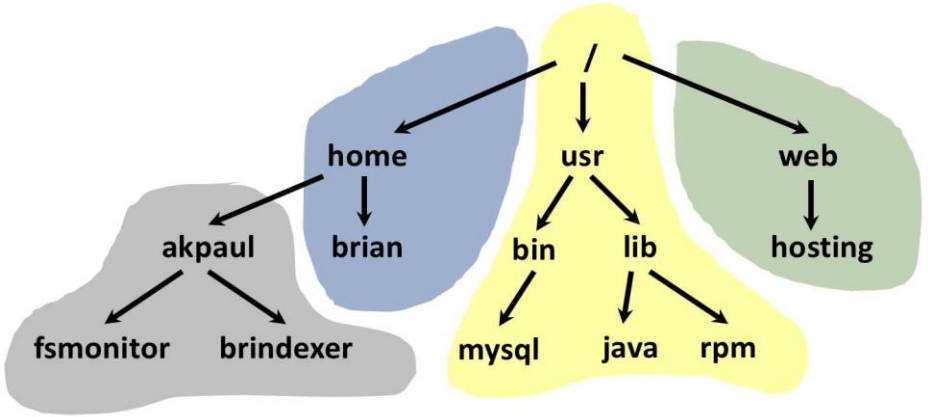
Bildquelle: siehe Literatur [1]

Wie funktioniert Metadaten-Indexing?

- Hierarchische Partitionierung
  - Spyglass, SmartStore, GIGA+, …
  - ➢ Schlechte Performance, Unbalancierte Trees

- Verwaltung einer Index Struktur in externer Datenbank
  - GUFI, Borg FS, Robinhood Policy Engine, …
  - ➢ Schwierig, konsistent zu halten

- Aktualisieren von Metadaten
  - Periodische Snapshots der Metadaten
  - Re-Indexing bei Änderungen
  - ➢ Aufwändig bei großen Dateisystemen



Bildquelle: siehe Literatur [1]

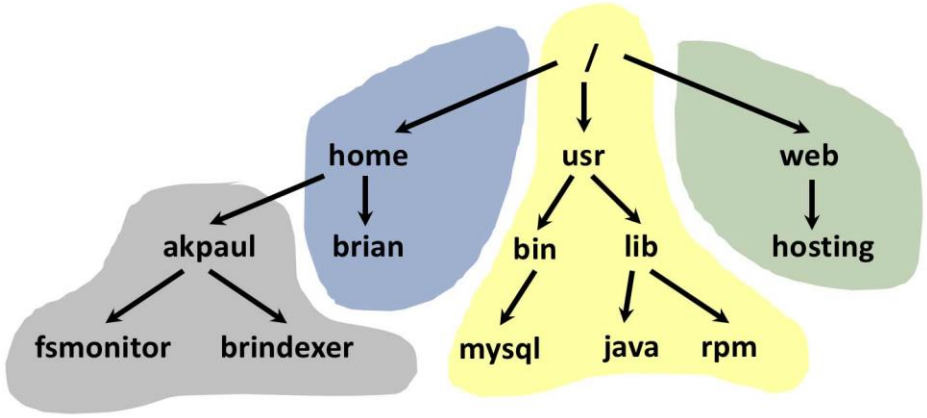
Brindexer

- Baut auf Lustre auf
- An Admins gerichtet
- Nutzt leveled Partitioning
- Speichert Metadaten in relationaler Datenbank
- Effizientes Re-Indexing mithilfe von Changelogs
- Bietet spezielles Query Interface

Balancierte Partitionierung

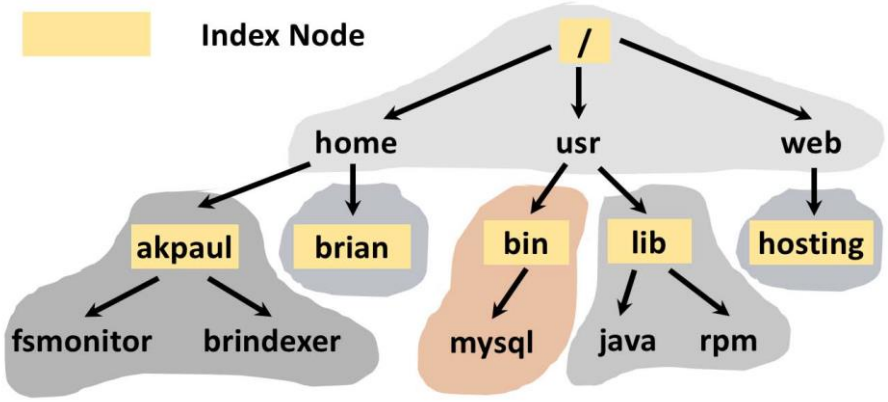
Hierarchical

- Verwendet zunächst File System Crawler
- Gleichmäßig große disjunkte Teilbäume
- Häufig nicht balancierte Bäumen



Leveled

- Indexknoten auf spezifischer Ebene erstellen
- Bäume sind balancierter als beim hierarchischen Verfahren



Bildquelle: siehe Literatur [1]

Sammeln von Metadata Änderungen

Snapshot-based

- Periodische Snapshots der Metadaten
- Diff bestimmen
- Re-Indexing des Diff

Changelog-based

- Re-Indexing von veränderten Dateien

Event ID	Type	Timestamp	Datestamp	Flags	Target FID	Parent FID	Target Name
11332885	01CREAT	22:27:47.308560896	2019.11.28	0x0	t=[0x300005716:0x626c:0x0]	p=[0x300005716:0xe7:0x0]	hello.txt
11332886	17MTIME	22:27:47.327910351	2019.11.28	0x7	t=[0x300005716:0x626c:0x0]		hello.txt
11332887	08RENME	22:27:47.416587265	2019.11.28	0x1	t=[0x300005716:0x17a:0x0]	p=[0x300005716:0xe7:0x0] s=[0x300005716:0x626b:0x0] sp=[0x300005716:0x626c:0x0]	hello.txt hi.txt
11332888	02MKDIR	22:27:47.421587284	2019.11.28	0x0	t=[0x300005716:0x626d:0x0]	p=[0x300005716:0xe7:0x0]	okdir
11332889	06UNLNK	22:27:47.438587347	2019.11.28	0x0	t=[0x300005716:0x626b:0x0]	p=[0x300005716:0xe7:0x0]	hi.txt

Lustre Changelog Ausschnitt

Quelle: siehe Literatur [1]

Verteilung von Event calls

Event Type	# Events		Event Type	# Events
CREAT	1,322,010		MKDIR	67,791
HLINK	8,841		SLINK	94,711
MTIME	10,098,485		UNLNK	750,480
RMDIR	59,841		RENME	97,227
SATTR	3,432,589		XATTR	164

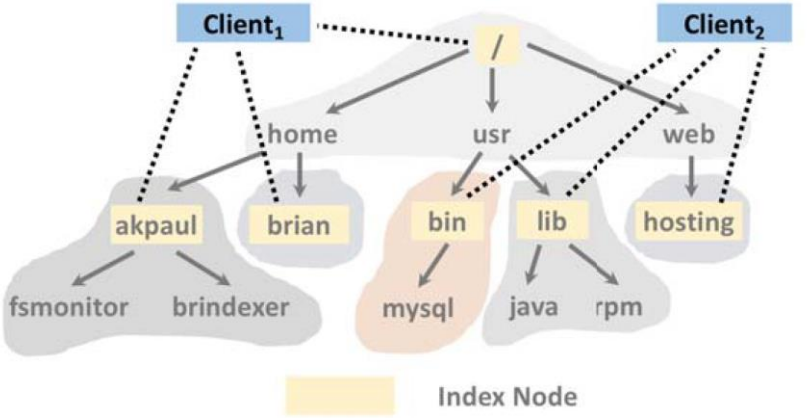
Snapshot von LANL

Bildquelle: siehe Literatur [1]

- 34 Mio Events pro Tag
- 10,5 Mio Dateien pro Tag betroffen
- 110.000 Verzeichnisse pro Tag betroffen

Brindexers Indexer

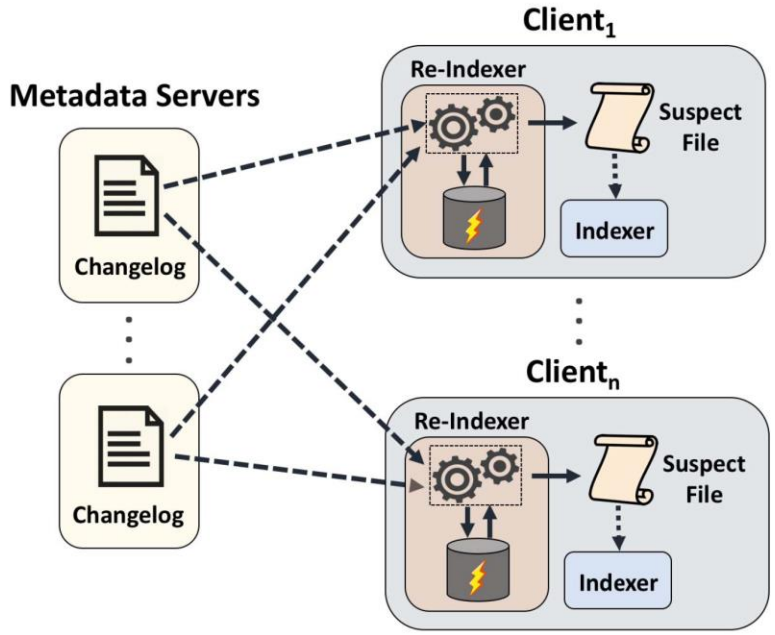
- Paralleles Indexing durch mehrere Clients
- Erstellt pro Index-Node eine fragmentierte Datenbank
- Geht rekursiv durch die Verzeichnisse beginnend bei den Index-Nodes
- Für jede Datei, wird dessen Inode einem Fragment zugewiesen
  - Fragment wird per MD5 Hash des Verzeichnis ermittelt



Bildquelle: siehe Literatur [1]

Brindexers Re-Indexer

- Verarbeitet Changelogs von Metadaten-Servern
- Nutzt LRU-Cache mit Parent FIDs zu absoluten Pfaden Mappings
- Erstellt *Suspect* Datei der modifizierten Verzeichnisse
- Filtert nur die neuesten Ereignisse und löscht die alten
- Periodisch wird die *Suspect* Datei an den Indexer übergeben und eine neue Datei begonnen

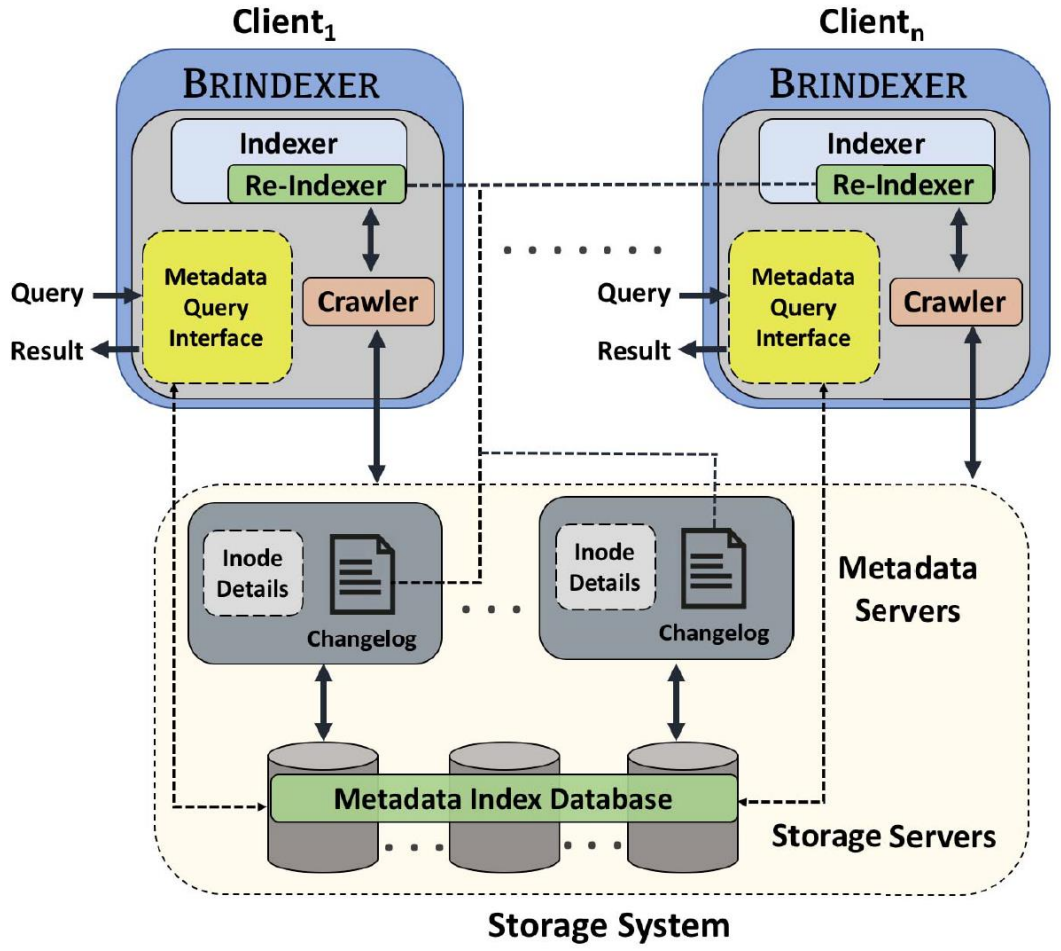


Bildquelle: siehe Literatur [1]

Brindexers Metadaten Query Interface

- Interagiert mit der Metadata Index Datenbank (RDBMS)
- Datenbank Fragmente machen Queries effizient
- RDBMS ist effizient mit Bulk I/O

- Problem bei kontinuierlichen I/O Streams
  - ➢ Brindexer schreibt nur periodisch
- Problem bei parallelem Input wegen Locks
  - ➢ Fragmentierung hilft, damit dies nicht zu häufig passiert



Bildquelle: siehe Literatur [1]

Aufbau der Auswertung

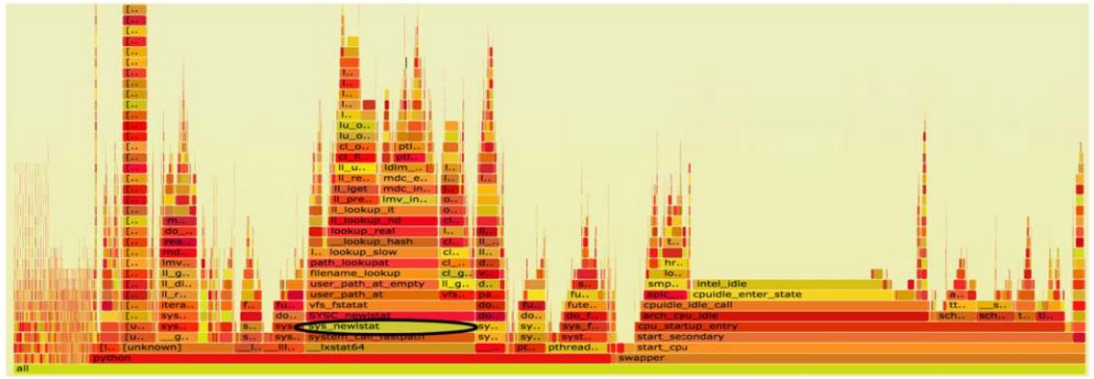
- Lustre System
  - 2 Clients; 4 MDS mit je 128GB MDT; 3 OSS mit je 3x 1,6 TB OST
  - Alle Knoten haben CentOS; AMD 64-Core 2,7GHz Prozessoren; 128GB RAM; 2,5TB SSD

- Test-Dateisysteme
  - **Flache Struktur**: wenige Ordner, durchschnittlich viele Dateien
  - **Verschachtelte Struktur**: viele Ordner (max. Tiefe 17), durchschnittlich wenige Dateien
  - Für beide Strukturen, verschiedene Mengen an Dateien (400K; 1,2M; 5,7M; 8,5M; 22M)

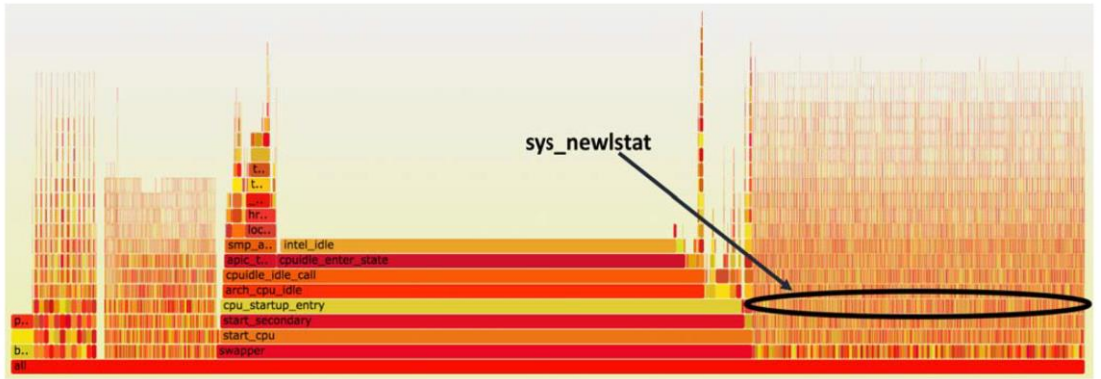
- Vergleich mit State-of-the-Art Indexern
  - GUFI
  - Lustres Standard Tool *lfs find*
  - Robinhood Policy Engine

Vergleich der System Calls

- Indexing von 1,2 Mio Dateien in hierarchischer Verzeichnisstruktur
- Die Breite jedes Blocks zeigt die Zeit an, die ein Systemaufruf auf der CPU verbringt



Brindexer



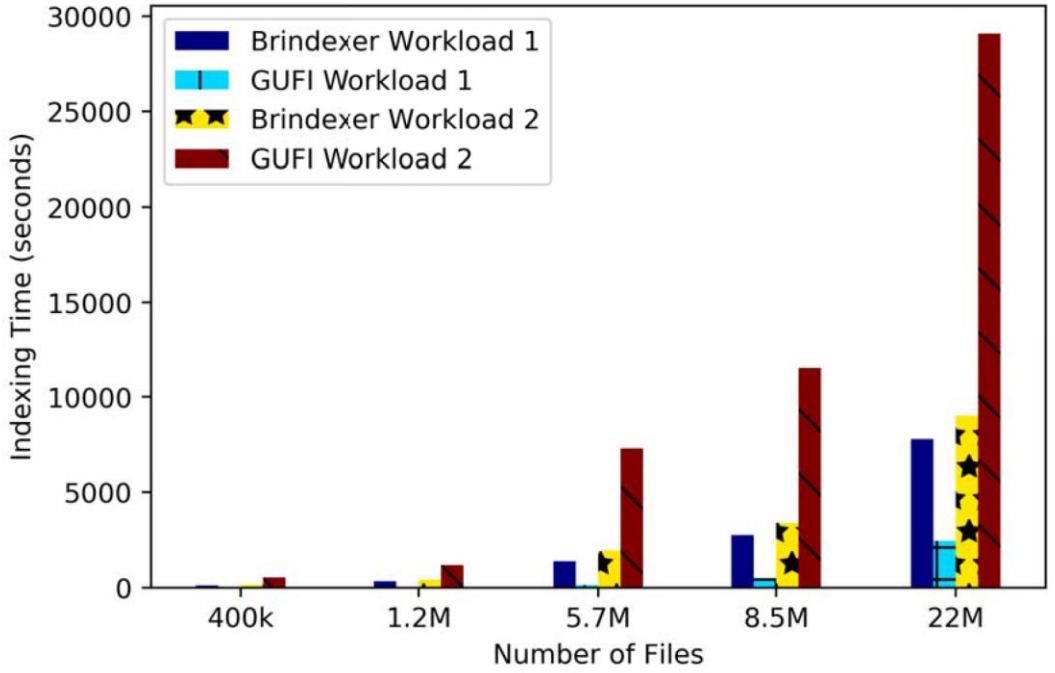
GUFI

Bildquelle: siehe Literatur [1]

➢Brindexer ist effektiver beim Sammeln der Inode informationen

Auswertung des Indexers

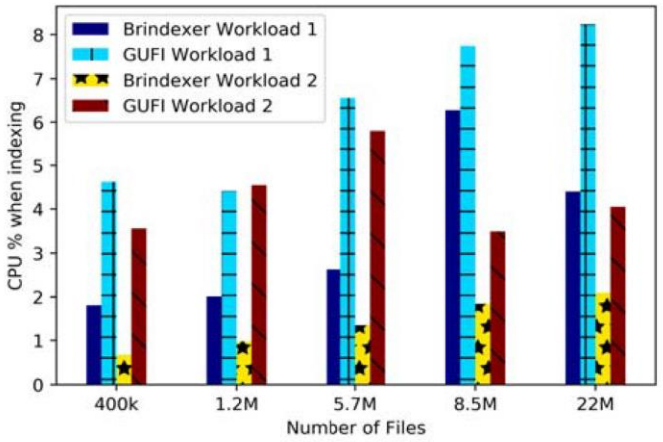
- GUFI schneidet bei flachen Verzeichnisstrukturen besser ab
- Brindexer schneidet bei hierarchischer Verzeichnisstrukturen besser ab
- Die Indizierungszeit von GUFI steigt exponentiell mit der Anzahl der Dateien



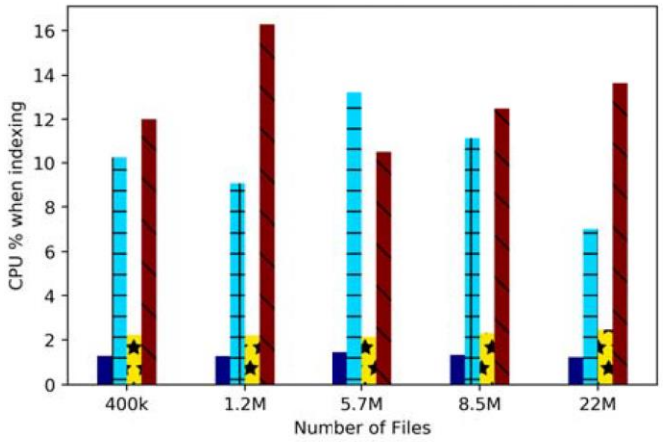
Indizierungszeit

Bildquelle: siehe Literatur [1]

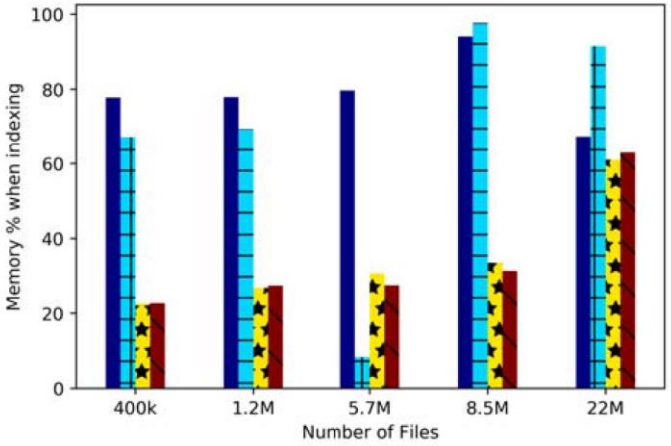
Indexing Systemauslastung



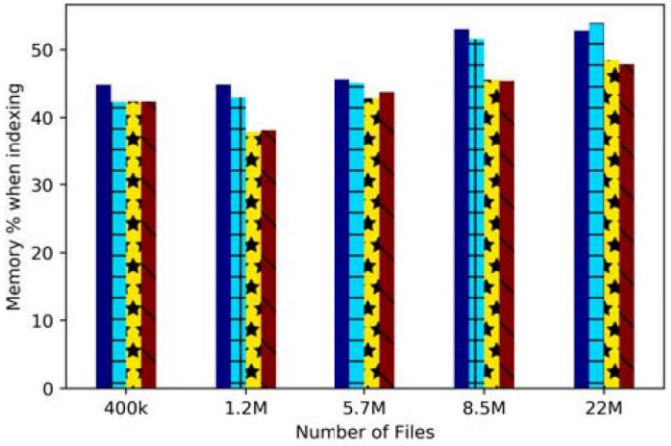
CPU% des Client



CPU% des MDS



Memory% des Client



Memory% des MDS

Bildquelle: siehe Literatur [1]

Auswertung des Re-Indexers

- Simulation von Änderungen durch ein Skript, das Zufallsereignisse auf 22 Mio Dateien erzeugt
- 766 Zufallsereignisse pro Sekunde pro MDS (Erstellen, Ändern, Entfernen)
- 28,7 % Verbesserung der Ereignisberichterstattungsrate mit LRU-Cache
- Beste Ressourcennutzung mit Cachegröße 5000 (nur 2,94% CPU und 62,4MB Speicher)

Cache Size (#fid2path)	CPU% on client	Memory (MB) on client	Events/sec reported by BRINDEXER
200	4.8	88.7	578
500	3.5	84.3	624
1000	2.98	75.6	659
2000	2.95	61.3	698
5000	2.94	62.4	734
7500	2.92	60.7	720

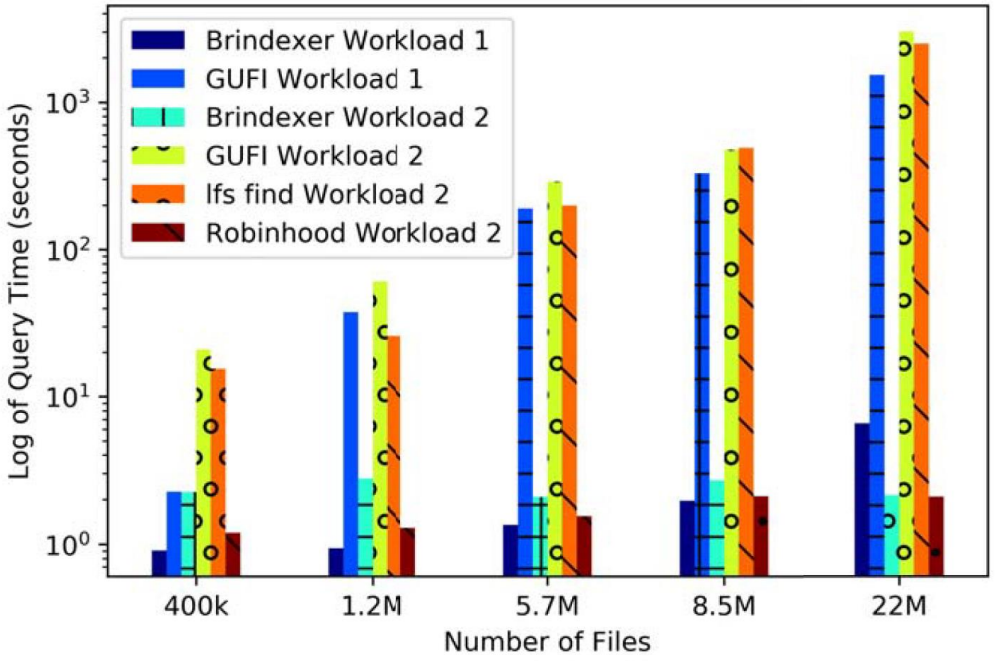
Brindexer Performance

Quelle: siehe Literatur [1]

Auswertung des Metadata Query Interface

- Finde Alle Dateien größer als 10MB
- Brindexer nutzt parallele Suche und eine fragmentierte Datenbank
- *lfs find* ist proportional langsam zu der Anzahl Index Knoten
- Robinhood nutzt eine externe Datenbank

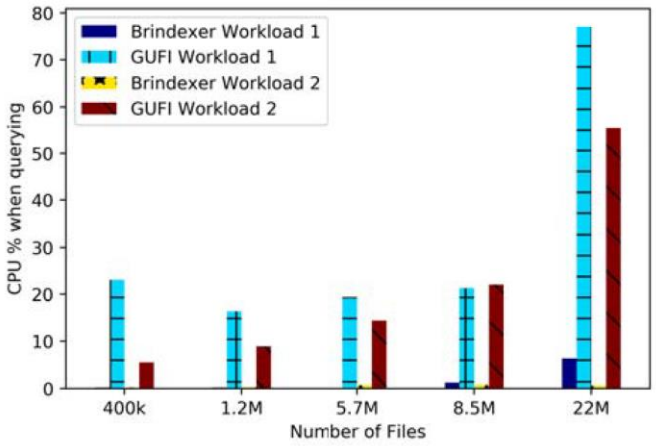
➢Brindexer ist ideal auf dem Dateisystem selbst



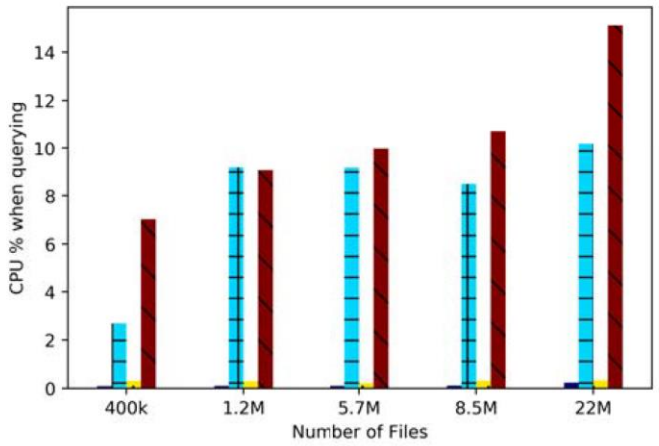
Query Zeit

Bildquelle: siehe Literatur [1]

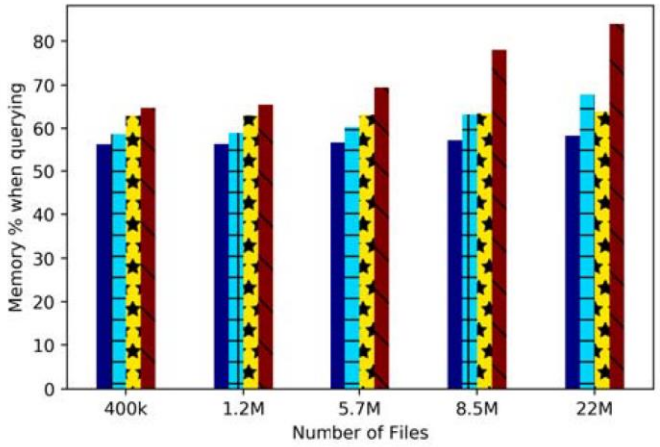
Metadata Query Interface Systemauslastung



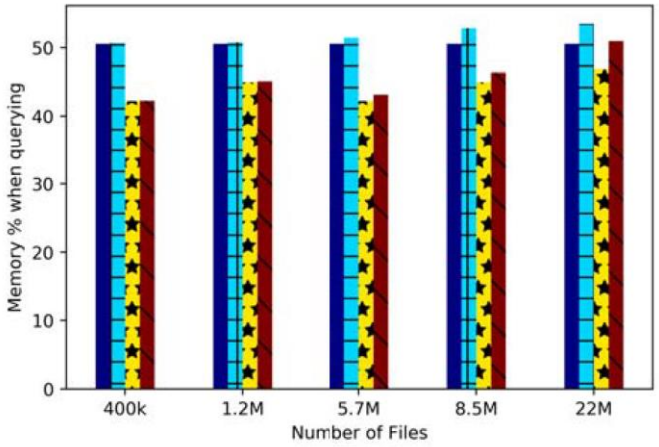
CPU% des Client



CPU% des OSS



Memory% des Client



Memory% des OSS

Bildquelle: siehe Literatur [1]

Zusammenfassung

- Brindexer ist ein verbesserter Metadaten-Indexer für HPC
  - Parallele, abgestufte Partitionierung
  - 2-Level-fragmentierte-Datenbank mit Verzeichnis Hashing
  - Changelog-basiertes Re-Indexing mit Chache
- Vergleich mit GUFI und Robinhood
  - Indizierungsleistung um 69% verbessert
  - Abfrageleistung um 91% verbessert
- Zukünftige Arbeiten sind geplant
  - Re-Indexer und Indexer verschmelzen um den Overhead der Suspect Datei zu beseitigen
  - Implementierung von Brindexer für andere HPC-Speichersysteme, z.B. BeeGFS und IBM Spectrum Scale

Quellen

[1] Paul, Arnab K., et al. "Efficient metadata indexing for hpc storage systems." *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 2020.

[2] Leibovici, T. (2015). Taking back control of HPC file systems with Robinhood Policy Engine.

[3] https://wiki.lustre.org/Lustre_Metadata_Service_(MDS)

[4] https://wiki.lustre.org/Lustre_Management_Service_(MGS)

[5] https://linuxhandbook.com/inode-linux/

[6] https://en.wikipedia.org/wiki/Copy-on-write

[7] https://www.man7.org/linux/man-pages/man7/xattr.7.html