Netzwerkspeicher und Dateisysteme

Jedes Dokument, jedes Bild, jedes Musikstück - kurz, jedes Datum mit dem wir auf Computern tagtäglich hantieren wird irgendwo gespeichert. In den meisten Fällen liegt es auf einem lokalen oder einem über das Netzwerk angebundenes Speichergerät. Aber wie werden Bits und Bytes nun wirklich auf Festplatte, SDD oder USB Stick bzw. auf Netzlaufwerken abgelegt? Dafür, dass man sich als Benutzer darüber keine Gedanken machen muss, sorgen Dateisysteme.

Diese Veranstaltung will einen Einblick in moderne Dateisysteme geben und möchte in einem zweiten Teil die Vorteile von großen Netzwerk-Speicherlösungen (NAS - Network Attached Storage) gegenüber einer am Rechner angeschlossenen USB Platte aufzeigen. Anhand von zwei großen NAS Herstellerlösungen wird die Funktionsweise von NAS Filern erklärt.

Wozu braucht man Dateisysteme?

- Benutzer muss sich nicht darum kümmern wie die Daten auf der Hardware abgelegt werden
- Prozesse und Benutzer greifen auf Dateien abstrakt über deren Dateinamen zu - Abstraktion für den Benutzer zwischen seiner Anwendung und den Daten die physikalisch auf der Platte liegen
- organisieren die Ablage von Dateien auf Datenspeichern
  - Dateien sind beliebig lange Folgen von Bytes und enthalten inhaltlich zusammengehörende Daten
  - Organisieren von Dateien in Katalogen (Verzeichnissen)
  - Wiederfinden von Dateien durch eine gewisse Logik in der Benamsung von Dateien und in der Ablage
- verwalten Dateinamen und Attribute (Metadaten) der Dateien
- bilden einen Namensraum
  - Hierarchie von Verzeichnissen und Dateien
- sind eine Schicht des Betriebssystems (==⇒ Systemsoftware)
  - Dateisystem greift auf (physische) Speicheradressen zu
- sollen wenig Overhead für Verwaltungsinformationen benötigen

Aufbau einer Festplatte

- ein oder mehrere Platten
- entsprechend ein oder mehrere Plattenarme mit den Schreib/Lese Köpfen (ober und Unterseite)
- Platten sind unterteilt in
  - Spur

    - wie bei einer Schallplatte, nur konzentrisch
  - Zylinder
    - übereinanderliegende Spuren
  - Sektoren
    - Abschnitte einer Spur
    - nehmen zum Mittelpunkt der Platte hin ab (?)
    - Präambel
    - Datenbist
    - ECC
  - Zonen
    - mehrere Zylinder bilden eine Zone mit gleicher Sektorenanzahl

Partitionierung von Festplatten

- aufteilen einer physikalischen Platte in mehere logische Platten
- Partitionen stellen sich für das Dateisystem wie eine „ganze“ Platte dar

Warum partitionieren?

- verschiedene Betriebssysteme auf einer Platte
- Trennung Daten und Betriebssystem
- Trennung Bereiche des OS, etwa loggingbereich, Homes, tmp, …
- eigener swap Bereich
- unterschiedliche Dateisysteme (Datenbanken, Datenaustausch, …)
- Backup
- Platte „verkleinern“ wenn die Gesamtkapazität für das OS zu groß ist (z.B. FAT 16 max. 2GB)
- Partitionen haben eine Magic-Number (*System ID*) die den Typ des dort verwendeten Dateisystems kennzeichnet

Partitionen für DOS, Windows und Linux

- prinzipiell max. 4 primäre Partitionen
- Erweiterung zu max. 3 primäre und eine erweiterte Partition
- erweiterte Partition Container für weitere Partitionen (Linux: 12 bei SATA, 60 bei IDE, Windows: beliebig viele)

andere Betriebssysteme, z.B. Solaris

- 8 logische Partitionen („slices“), von denen eine (slice 2) als pseudopartition die gesamte Platte darstellt

Speicherung von Daten

- Dateisysteme adressieren **Cluster** und nicht die **Blöcke** des Datenträgers
- in der Literatur werden **Cluster** auch manchmal als **Zonen** (*Zones*) oder **Blöcke** (*Blocks*) bezeichnet. Das fürt zu Verwechslungen mit den Blöcken (Sektoren) auf Festplattenebene
- Die Größe der Cluster ist wichtig für die Effizienz des Dateisystems
  - Je kleiner die Cluster:
    - Steigender Verwaltungsaufwand für große Dateien
    - Abnehmender Kapazitätsverlust durch interne Fragmentierung
  - Je größer die Cluster:
    - Abnehmender Verwaltungsaufwand für große Dateien
    - Steigender Kapazitätsverlust durch interne Fragmentierung
- Clustergröße kann oft beim Anlegen des Dateisystems festgelegt werden.
- Dateien brauchen oft mehr als nur einen Cluster auf Platte
- Daher verschiedene Konzepte um Daten/Dateien zu speichern:
  - Kontinuierliche Speicheung
  - Verkettete Speicherung
  - indiziertes Speichern

Kontinuierliche Speicherung

- Datei wird in Blöcken mit aufsteigenden Blocknummern gespeichert
- Nummer des ersten Blocks und Anzahl der Folgeblöcke muss gespeichert werden

Vorteile

- Zugriff auf alle Blöcke mit minimaler Positionierzeit des Schwenkarms
- Schneller direkter Zugriff auf bestimmter Dateiposition
- Einsatz z. B. bei Systemen mit Echtzeitanforderungen

Probleme

- Finden des freien Platzes auf der Festplatte (Menge aufeinanderfolgender und freier Plattenblöcke)
- Fragmentierungsproblem (Verschnitt: nicht nutzbare Plattenblöcke; siehe auch Speicherverwaltung)
- Größe bei neuen Dateien oft nicht im Voraus bekannt
- Erweitern ist problematisch
  - Umkopieren, falls kein freier angrenzender Block mehr verfügbar

Variation

- Unterteilen einer Datei in Folgen von Blöcken (Chunks, Extents)
- Blockfolgen werden kontinuierlich gespeichert
- Pro Datei muss erster Block und Länge jedes einzelnen Chunks gespeichert werden

Problem

- Verschnitt innerhalb einer Folge (siehe auch Speicherverwaltung: interner Verschnitt bei Seitenadressierung)

Verkettete Speicherung

- Cluster einer Datei sind verkettet
- Beispiel: Commodore 64
- die ersten zwei Bytes bezeichnen Spur- und Sektornummer des nächsten Blocks
- wenn die nächste Spurnummer Null ist -> letzter Block
- eine Datei kann „wachsen“

Probleme

- Ein Cluster beinhaltet immer auch den Zeiger auf den nächsten Cluster. Damit geht Speicherplatz für Nutzdaten verloren
- tritt mitten in der Datei ein Fehler auf und ist der Zeiger auf den nächsten Block fehlerhaft, so ist der Rest der Datei verloren.
- Direkter Zugriff auf eine bestimmte Dateiposition (*seek*) aufwändig (man muss sich von Cluster zu Cluster hangeln)

Indiziertes Speichern

- spezieller Cluster enthält die Clusternummern der Daten-Cluster einer Datei
- Problem dabei: feste Anzahl möglicher Daten-Cluster im Index-Cluster, für größere Dateien Erweiterung nötig
- Lösung:

mehrstuftige Indizierung

- mehrere verknüpfte Index-Cluster ermöglichen es auch große Dateien zu adressieren.
- Nachteil: je nach Dateigröße müssen neben den Daten-Clustern mehrere Index-Cluster geladen werden

FAT - File Allocation Table

- FAT verwendet *verkettete Speicherung* bei der die Zeiger nicht in den Daten-Clustern liegen sondern in einer speziellen Tabelle mit fester Größe
- damit: Daten-Cluster ist komplett für Daten nutzbar
- verschiedene Versionen (Nummer gibt länge in Bits der Einträge in der FAT Tabelle an):
  - *FAT12* 1980
    - wird heute noch für Win/DOS Disketten benutzt
    - Dateinamen 8.3
  - *FAT16* 1983
    - 2^16 = 65524 Cluster adressierbar, 12 davon sind reserviert.
    - Clustergröße: 512 Bytes - 15kB
    - Windows NT: 64kB Clustergröße, 4GB Dateigröße
    - MS-DOS und Win9x 2GB Dateigröße
    - wird heute noch für mobile Datenträger mit Speicher < 2GB verwendet
    - Dateinamen 8.3
  - *FAT32* 1997
    - 2^32 = 4.294.967.295 Cluster adressierbar
    - jeweils 4 Byte reserviert
    - dadurch: 2^28 = 268.435.456 adressierbar
    - Clustergröße 512 Bytes bis 32kB
    - Dateinamen 255 Zeichen
    - max. Dateigröße 4GB
    - wird heute noch für mobile Datenträger mit Speicher < 2GB verwendet
  - *VFAT* (Virtual FAT) 1997
    - Erweiterung für FAT12, FAT16 und FAT32
    - damit auch für ältere FATs Dateinamen mit 255 Zeichen und Unicode möglich

Nachteile

- mindestens ein zusätzlicher Daten-Cluster (die Tabelle) muss geladen werden…
- …oder zur Effizienzsteigerung im Speicher gecached werden
- FAT Tabelle enthält *alle* Datencluster, auch solche, die nicht benötigt werden
- Suche nach bestimmter Position in einer Datei (*seek*) ineffizient
- häufiges Positionieren des Schreib-/Lesekopfes bei verstreuten Daten-clustern -> Defragmentierung notwendig

Aufbau

- für jeden Cluster existiert ein Eintrag in der Dateizuordnungstabelle mit folgenden Informationen über den Cluster:
  - Cluster ist frei oder das Medium an dieser Stelle beschädigt
  - Cluster ist von einer Datei belegt und enthält die Adresse des nächsten Clusters, der zu dieser Datei gehört bzw. ist der letzte Cluster der Datei

Bereiche im FAT Dateisystem

- *Bootsektor* ausführbarer x86 Maschinencode (lädt das OS) und Informationen über das Dateisystem:
  - Blockgröße des Speichermediums (512, 1.024, 2.048 oder 4.096 Bytes)
  - Anzahl der Blöcke pro Cluster
  - Anzahl der Blöcke (Sektoren) auf dem Speichermedium
  - Beschreibung des Speichermediums
  - FAT-Version
- *reservierter Bereich* der ggf. vom Bootmanager verwendet wird
- *FAT 1* enthält alle reservierten und freien Cluster
  - Konsistenz dieser Tabelle ist elementar, deshalb:
- *FAT 2* zweite Kopie der FAT Tabelle
- *Stammverzeichnis* im Wurzelverzeichnis ist jede Datei und jedes Verzeichnis repräsentiert.
  - FAT12 und FAT16: feste größe und Position (gleich hinter FAT2)
  - FAT32 beliebige Größe und Position im Dateisystem
- *Datenbereich*

NTFS - New Technology File System

- eingeführt mit Win NT 3.1
- Max Dateigröße 16TB
- Max Partitionsgröße 256 TB
- Max. Länge Dateinamen 255 Zeichen
- Zugriffsrechte auf Dateien und Verzeichnisse
- Dateinamen Unicode Zeichen (ausser 0 und /)
- Clustergröße 512 Bytes - 64kB
- Verschiedene Versionen, aktuell 3.1, abwärtskompatibel
- Features:
  - transparente Verschlüsselung (3DES und AES (ab XP)) ab Version 2.x, EFS (Encrypting File System) ab 3.x

  - transparente Kompression
  - Quota ab Version 3.x
  - Hardlinks ab Version 3.x
  - *Reparse Points* (Analysepunkte) ab 3.x
    - spezielles Attribut
    - ermöglicht spezielle Treiber für spezielle Dateien
    - wird eine Datei mit so einem Attribut gefunden wird diese Datei zum nochmaligen Parsen an die einzelnen im OS geladenen *file system filter driver* übergeben der ggf. speziellen Code ausführen kann. Damit ist auch ext. Verschlüsselungssoftware möglich
  - *Alternate Data Streams* (ADS) vergleichbar mit resource forks (Infos z.B. über Downloads über Web, …)
  - *Volume Shadow Copy* = Snapshots ab Version 3.x
  - verwendet Journaling für Metadaten, seit Vista Transactional NTFS (TxF) -> atomare Operationen möglich, allerdings ist das von Microsoft wieder deprecated
- Dateien werden zusätlich mit 8.3 Namen abgespeichert
- Problem: muss eindeutig sein
- Lösung:
  - alle Sonderzeichen und Punkte bis auf den letzten löschen
  - alle Kleinbuchstaben -> Großbuchstaben
  - erste 6 Buchstaben werden beibehalten, danach eine ~1 vor dem Punkt
  - ersten drei Zeichen nach dem Punkt bleiben, Rest gelöscht
  - existiert eine Datei mit dem Namen wird das ~1 hochgezählt (~2, ~3, …)

Aufbau

- eine Hauptdatei, die **Master File Table** (MFT)
- enthält Referenzen welche Blöcke zu einer Datei gehören
- enthält Metadaten der Dateien (größe, Datum Erstellung, Änderung, Freigabe, Dateityp, und ggf. Inhalt)
- kleine Dateien werden direkt in der MFT gespeichert
- Beim Formatieren wird default 12,5% der Partitionsgröße für die MFT reserviert
- reicht der Platz nicht wird weiterer Speicher im Dateisystem verwendet -> MFT wird fragmentiert

klassische Unix Dateisysteme

- klassische Unix Dateisysteme verwendet *indiziertes Speichern* mit mehrstufiger Indizierung

- Jede Datei besteht somit aus mindestens einem oder mehreren Inodes und den eigentlichen Daten

Aufbau System V File System

- *Bootblock* erster Block, Bootmanager
- *Superblock*
- *Inodes*
- *Daten-Cluster*

Aufbau (ext2/3)

- Berkeley FFS und ext2/3 sind sich ähnlich:
  - *FFS*: Zylindergruppen (Menge aufeinanderfolgender Zylinder (häufig 16))
  - *ext2/3*: Blockgruppen unabhängig von Zylindern
- *FFS*: Daten werden möglichst innerhalb einer Zylindergruppe gehalten
- *FFS*: *Cylinder Group Block* enthält freie Inodes und Daten-Cluster
- *ext2*: Cluster werden in Blockgruppen gleicher Größe zusammengefasst (max. 8x Clustergröße in Bytes, Beispiel: Cluster 1024bytes -> Blockgruppe max. 8192 Cluster groß)
- Vorteil Blockgruppen: Inodes liegen physikalisch nahe an den Clustern die sie adressieren
- *Bootblock* erster Block, Bootmanager
- *Superblock*
  - Informationen zum Dateisystem (Anzahl Inodes und Blöcke gesamtes System, Anzahl Inodes und Blöcke innerhalb Blockgruppen, wann eingebunden, ob korrekt ausgeworfen, Version, letzte Änderungen, welches OS usw.
  - liegt 1024 Bytes hinter Anfang des Gerätes
- *Blockgruppen*:
  - *Superblock* jede Blockgruppe enthält eine Kopie des Superblocks
  - *Deskriptortabelle*
    - Die Clusternummern des Block-Bitmaps und des Inode-Bitmaps
    - Die Anzahl der freien Cluster und Inodes in der Blockgruppe
  - *Block Bitmap* und *Inode Bitmap*
    - Sie enthalten Informationen über die Cluster- und Inode-Belegung in der Blockgruppe
    - Diese Bitmaps sind ein Abbild der freien und belegten Cluster und Inodes
  - *Inode Tabelle*
  - *Daten-Cluster*

HFS +

- Nachfolger von HFS
- 32bit Blockadressen (anstatt 16bit HFS)
- Unicode Dateinamen
- Dateinamen 255 UTF-16 Zeichen
- verwendet logische Sektoren von 512 bytes (default)
- diese Sektoren werden zu allocation blocks (2^32) zusammengefasst welche einen oder mehrere logische Sektoren beinhalten können
- besteht aus 9 Strukturen
  1. Sektoren 0 und 1 - **boot block**
  2. Sektor 2 **volume header** (Infos zum Dateisystem wie größe der allocation blocks, …)
  3. **allocation file** mit Liste der benutzten und unbenutzten Blöcke. Ist eine (unsichtbare) „echte“ Datei
  4. **catalog file** b-tree welcher alle Dateien und Verzeichnisse auf dem Laufwerk enthält. Ein Eintrag ist 8kb groß
  5. **extents overflow file** b-tree jeder Eintrag zu einer Datei im **catalog file** kann 4 extensions pro fork speichern. Jede weitere wird im **overflow** gespeichert. 4kb pro Eintrag
  6. **attributes file** kann drei jeweils 4kb große record-Typen speichern:
     - **inline data attribute records**
     - **fork data attribute records**
     - **extension attribute records**
  7. **startup file** ähnlich dem **boot block** aber für Betriebssysteme die keinen HFS oder HFS+ Support haben
  8. **alternate volume header**
  9. der letzte Sektor ist Apple vorbehalten: *used during Apple's CPU manufacturing process* (Apple Doku)
- n-forked files, verwendet werden aber nur resource forks:
  1. eine Art unsichtbare „Huckepack-Datei“
  2. können beliebige Daten enthalten, etwa
     - Text-Dokumente: Text im data fork, Bilder im resource fork
     - meistens verwendet für Metadaten der Datei (Autor, Notizen, Tags, Icon-Bitmap, extended Attributes)
  3. Data-forks werden wie klassische Dateien angesprochen (open, seek, datenstream lesen, close)
  4. ressource-forks werden wie Datenbanken angesprochen (open mit Angabe welche Ressource benötigt werden, OS liefert das Ergebnis)
  5. Ressourcen haben einen 4-ASCII Zeichen grossen Identifier. Beispiele:

    - DITL Dialogfenster
    - PICT Bilder
    - `snd` Audiodaten
    - CODE ausführbaren Programmcode
    - WDEF Window definition
    - MDEF Menu definition
  6. Probleme mit Abspeichern auf nicht-HFS+ (etwa auf Netzwerklaufwerken)
  7. Resource forks auf nicht-HFS+ werden in versteckten Verzeichnissen abgelegt
- unterstützt im Gegensatz zu anderen Unixoiden Betriebssystemen keine *sparse files*

Journaling

- zusätzlich zum Schreiben der Daten und Metadaten wird Protokoll geführt
- Möglichkeiten: *Metadaten-Journaling* und *vollständiges Journaling*

Metadaten Journaling

- Das Journal enthält nur Änderungen an den Metadaten
  - Änderungen werden direkt in das Dateisystem geschrieben
  - Nur die Konsistenz der Metadaten ist nach einem Absturz garantiert
- Vorteil: Konsistenzprüfungen dauern nur wenige Sekunden
- Nachteil: Datenverlust durch einen Systemabsturz ist weiterhin möglich
- Optional bei ext3/4 und ReiserFS
- NTFS bietet ausschließlich Metadaten-Journaling

vollständiges Journaling

- Änderungen an den Metadaten und alle Änderungen an Clustern der Dateien werden ins Journal aufgenommen
- Vorteil: Auch die Konsistenz der Dateiinhalte ist garantiert
- Nachteil: Alle Schreiboperation müssen doppelt ausgeführt werden
- Optional bei ext3/4 und ReiserFS

Ordered Journaling

- kompromiss aus beiden, wird meistens in Unix Dateisystemen verwendet:
- Das Journal enthält nur Änderungen an den Metadaten
- Dateiänderungen werden erst im Dateisystem durchgeführt und danach die Änderungen an den betreffenden Metadaten ins Journal geschrieben

- Vorteil: Konsistenzprüfungen dauern nur wenige Sekunden und hohe Schreibgeschwindigkeit wie beim Metadaten-Journaling
- Nachteil: Nur die Konsistenz der Metadaten ist garantiert
  - Beim Absturz mit nicht abgeschlossenen Transaktionen im Journal sind neue Dateien und Dateianhänge verloren, da die Cluster noch nicht den Inodes zugeordnet sind
  - Überschriebene Dateien haben nach einem Absturz möglicherweise inkonsistenten Inhalt und können nicht mehr repariert werden, da die alte Version nicht gesichert wurde
- Beispiel: XFS, JFS, Standard bei ext3/4 und ReiserFS

Copy on write

- aktuell modernstes Konzept
- Schreibzugriffe werden immer in freie Cluster geschrieben
- keine Daten werden überschrieben
- Nach abgeschlossenem Schreibvorgang werden Metadaten aktualisiert und auf neuen Cluster verlinkt
- ist dies abgeschlossen werden ggf. die Cluster freigegeben wo die Datei vorher gelegen hat
- Dann wird der Superblock aktualisert und zeigt auf die neue Metadaten/Inode Tabelle
- Vorteile:
  - immer konsistenter Zustand
  - einfache Implementierung von Snapshots:
    - bei Schreiboperationen wird der alte Superblock gesichert und zeigt damit auf die „alten“ Daten

ZFS

- Beispiel für copy-on-write Dateisystem
- ZFS ist nicht nur ein Dateisystem, bildet kompletten Layer von Plattenverwaltung (Volume Management - RAID-Gruppen) bis hin zu Freigaben über NFS und SMB ab.

technische Daten

- 128bit Dateisystem, theoretisch also Adressbereich von 2^128 (und damit 1.84*10^19 mal mehr als z.B. *btrfs*)
- max Volume Size 2^78 bytes
- max Dateigröße 2^64 bytes

- max Anzahl an Dateien 2^48
- max Dateilänge 255 ASCII-Zeichen (oder entspr. weniger Unicode Zeichen)

Volume Management

- physikalische Festplatten werden in sogenannten pools verwaltet
- ZFS „mag“ keine Hardware RAID Controller da optimierungen nicht greifen
- Redundanz:
  - mirror
  - RAID-Z wie RAID5 aber jeder Block ist sein eigener Stripe. 1 Platte kann ausfallen
  - RAID-Z2 wie RAID 6, 2 Platten können ausfallen
  - RAID-Z3 3 Platten können ausfallen
  - Spare Disks (werden automatisch eingebunden wenn Platte ausfällt)
- pools können beliebige (zfs) Dateisystemen enthalten
- pools können virtuelle Platten (zdefvs) zur Verfügung stellen
- pools können erweitert werden. zfs Dateisysteme werden dann einfach größer

Dateisystemn

- Verschlüsselung
- Komprimierung
- Snapshots (read only)
- Clones (r/w snapshot Kopie)
- Deduplizierung (RAM intensiv! Erfolgt online)
- NFSv4 ACLs
- NFS Freigaben
- SMB Freigaben (Solaris)
- snapshots send/recieve ermöglicht Konzepte wie räumlich getrenntes clustering

Datenintegrität

- jeder Datencluster wird mit einer Checksumme versehen die an einer anderen Stelle gespeichert und verzeigert wird
- dies erfolgt hoch bis zum Superblock
- Damit ist die Checksummer jedes Clusters in seinem Eltern-Cluster gespeichert
- Bei *jedem* Zugriff (Daten, Metadaten) wird die Checksumme berechnet und mit der gespeicherten verglichen
- Stimmt sie überein werden die Daten ans System übergeben

- Stimmt sie nicht werden die Daten aus der Zweitkopie (mirror, RAID Redundanz) gelesen.
- Wenn diese stimmen wird der Block mit den korrekten Daten aktualisiert
- Hat man nur eine Platte gibt es die Möglichkeit per Parameter (`copies=2`) mehrere Kopien der Daten anzulegen
- es gibt kein fsck, aber mit ``zfs scrub```kann man im Hintergrund und online alle Daten (Metadaten, Daten) auf ihre Integrität checken lassen

Performance Optimierung

- RAM Cache wie andere Dateisysteme auch
- L2ARC
  - Lese Cache der auf SSDs ausgelagert werden kann
  - hilft auch bei dedeuplizierung falls die dedup-Tabelle auf dem Medium Platz hat
  - falls die Platte kauptt geht -> Lese Zugriffe normal über Platte
- ZIL (ZFS Intention Log)
  - Schreib Cache der auf SSDs ausgelagert werden kann
  - Jede Schreiboperation wird in einer Art Journal auf SSD geschrieben
  - Je nach Auslastung wird dies zu einem späteren Zeitpunkt auf Platte abgelegt.
  - Schreiboperationen werden durch schnellere Medien (SSDs) beschleunigt und erst später auf langsamere Platte geschrieben
  - Problematisch bei Ausfall, deshalb mirror!

Plattformen

- Solaris, OpenSolaris, OpenIndiana
- versch. BSD Varianten (DragonFly BSD, NetBSD, FreeBSD, OS X, MidnightBSD, PC-BSD)
- NAS OS Distributionen wie FreeNAS, ZFS-Guru, NAS4Free, NexentaStor, EON NAS und andere
- Linux (FUSE, LLNL Implementierung, ...)

Fragmentierung und Defragmentierung

- in einem Cluster wird eine Datei gespeichert
- ist sie größer als das Cluster wird sie auf weitere Cluster aufgeteilt
- wenn Cluster hintereinander liegen -> alles gut
- liegen logisch zusammengehörende Cluster auf anderen Plattenbereichen -> Fragmentierung
- Fragmentierung spielen auf SSDs keine Rolle

Warum defragmentieren, wann macht es keinen Sinn

- durch Plattenkopfbewegungen Zeitverlust beim Zugriff
- Ziel: Daten-Cluster sollen möglichst nahe beieinander liegen um sie in einem Stream lesen zu können
- manche OSse cachen nichts oder wenig, hier machen Zugriffszeiten sehr viel aus

Aber:

- bei Multitasking OS:
  - Daten können selten am Stück gelesen werden da es auch andere Anwendungen/Tasks gibt die auf Platte zugreifen
  - Deshalb wird viel mehr gecached, vor allem Daten auf die häufig zugegriffen wird ->
  - Chance, dass Daten im Cache liegen anstatt auf Platte
- Deshalb bringt defragmentierung unter Unix nicht viel auch wenn die Daten auch dort prinzipiell fragmentiert über die ganze Platte verstreut liegen

- HSM (Hierarchical Storage Management)
  - eigentlich eine Technik um oft und wenig genutzte Daten auf unterschiedlichen Medien zu speichern (SSD, schnelle Platte, langsame Platte, Tape Drive)
  - Beispiele: Oracle SAM-QFS, IBM TSM (Tivoli Storage Manager), EMC Legato DiskXtender, Novell Dynamic Storage Technology (DST), ...
- ReFS (Microsoft, Windows Server 2012, soll NTFS ablösen)
- Tagsistant (Linux, FUSE)
  - semantisches Dateisystem welches keine klassische Verzeichnishierarchie verwendet und ausschließlich auf *Tags* basiert
- WinFS (Microsoft, eingestellt, **kein** eigenständiges Dateisystem, baut auf NTFS auf und bildet im Prinzip eine Schicht zum Auffinden/Verwalten von Dateien)