HPI Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Datenbanksysteme II

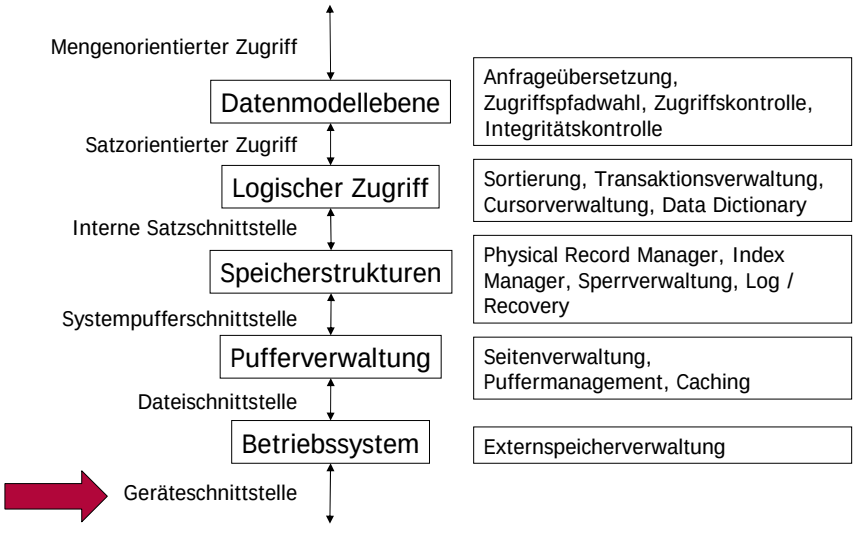
Physische Speicherstrukturen

(Kapitel 11)

18.4.2007

Felix Naumann

Zoom in die interne Ebene: Die 5-Schichten Architektur



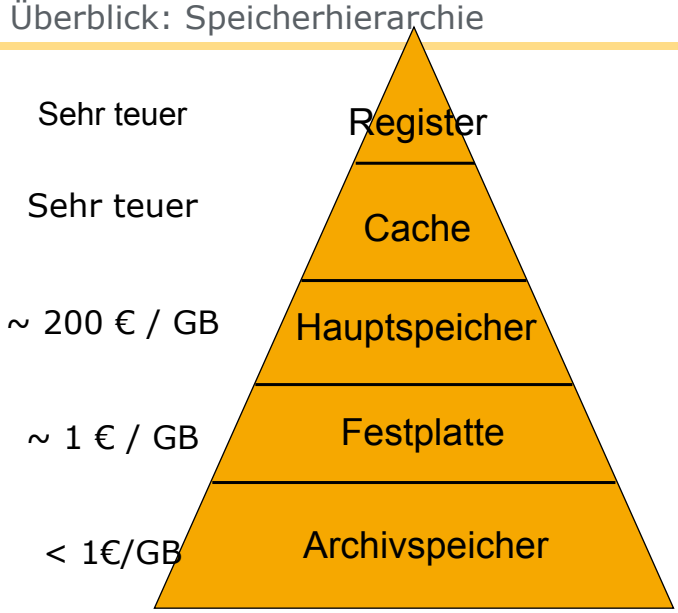
Übersicht

3

- Speicherhierarchie
- Disks
- Effiziente Diskoperationen
- Zugriffsbeschleunigung
- Diskausfälle
- RAID 0 – 6

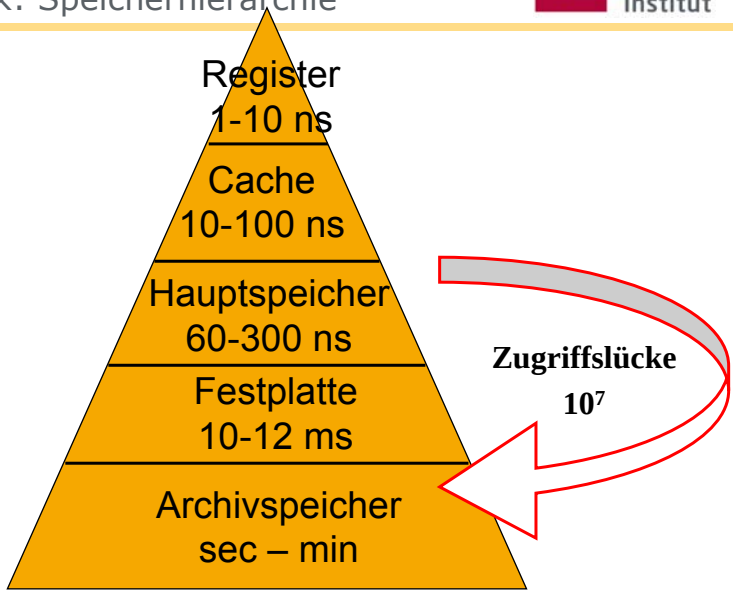
Überblick: Speicherhierarchie

4



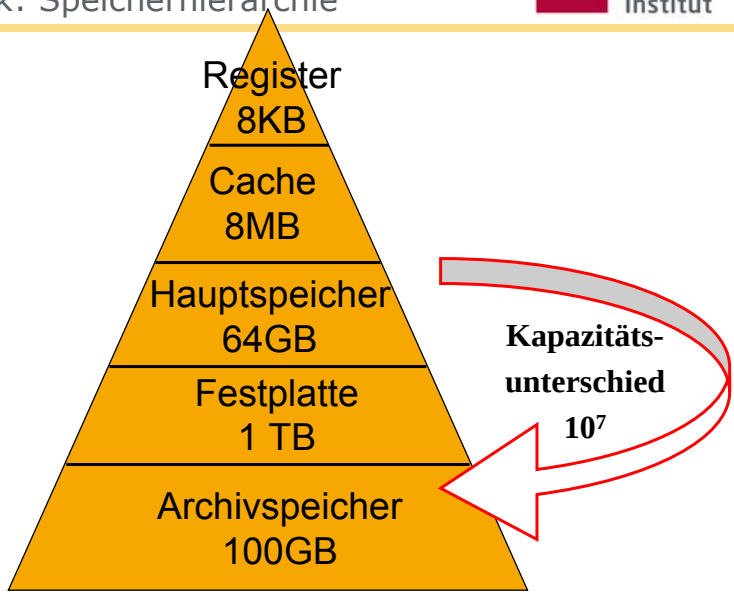
Überblick: Speicherhierarchie

5



Überblick: Speicherhierarchie

6



Cache

7

Allgemein: Cache Speicher vermindern Zugriffslücke zwischen Medien.

Puffern Daten der jeweils darunter liegenden Ebene

- On-board Cache
  - Auf Prozessorchip
- Level-2 Cache
  - Auf eigenem Chip
- Kopien bestimmter Speicherorte des Hauptspeichers
- Verdrängungsstrategie
  - Bei Verdrängung gegebenenfalls Daten im Hauptspeicher aktualisieren
- Write-through bei Mehrprozessormachinen
  - Jeder Prozessor hat eigenen Cache
  - Shared Memory

Cache – Lokalität des Zugriffs

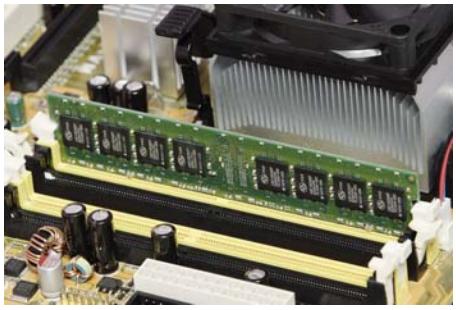
8

- Annahme: Typische Anwendungen können einen Großteil der Zugriffe (> 90%) mit Daten aus Cache beantworten
- Ohne Lokalität ist Cache nutzlos
- Cache-Hit-Ratio
  - $\frac{\text{\# of blocks requested in cache}}{\text{\# of blocks requested}}$

Hauptspeicher

Vorstellung: Sämtliche Aktionen eines Rechners geschehen im Hauptspeicher.

- Annahme: Random Access
  - Jedes Byte kann zu gleichen Kosten gelesen werden.



Virtueller Speicher

Jede Anwendung verwaltet einen virtuellen Adressraum.

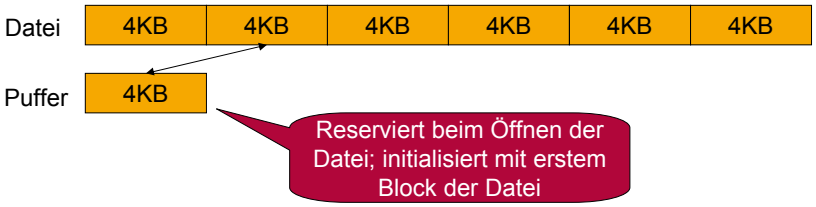
- Kann größer als tatsächlich verfügbarer Hauptspeicher sein
- 32-bit = 2^{32} Adressen = 4 GB
- Daten werden auf Disk ausgelagert
- Seitenweise Lesen und Schreiben
  - Verwaltet durch Betriebssystem
- Datenbanken verwalten Daten auf Festplatte selbst!
- Für Main-memory DBMS jedoch relevant
  - Wachsender Nichen-Markt
  - Sinnvoll bei kleinen, spezialisierten Datenbanken

Sekundärspeicher: Festplatten

- Nicht nur Festplatten; auch optische (read-only) Speicher
- Im Wesentlichen Random Access:
  - Zugriff auf jedes Bit kostet gleich viel
  - Aber: Erst einmal hinkommen!
- Halten Daten aus Cache / virtuellem Speicher
- Halten Daten aus Dateisystem
- Operationen
  - Disk-read
  - Disk-write
  - Beides: Disk-I/O

Festplatten - Puffer

- Hauptspeicher puffert Teile von Dateien
  - In Blockgröße (z.B. 4KB)



- DBMS verwaltet Blöcke selbst!
- Dauer für Schreiben oder Lesen eines Block: 10-30 ms
  - Entspricht viele Millionen Prozessoranweisungen
  - I/O Zeit dominiert Gesamtkosten
  - Deshalb am besten: Block sollte bereits im Speicher sein!

Tertiärspeicher: Magnetbänder

13

- Viele Terabyte an Verkaufsdaten
- Viele Petabyte Sattelitenbeobachtungsdaten
- Festplatten ungeeignet
  - Zu teuer (Wartung, Strom, Maschinen)
- Tertiärspeicher
  - I/O Zeiten wesentlich höher
  - Kapazitäten wesentlich höher (und billiger)
- Kein Random Access
  - Zugriffszeiten hängen stark vom jeweiligen Datensatz ab

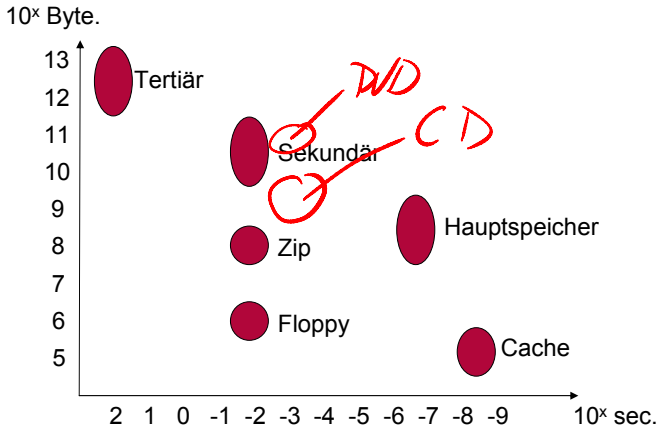
Tertiärspeicher

14

- Ad-hoc Speicherung auf Magnetbändern
  - Magnetbandspulen
  - Kassetten
  - Von Menschenhand ins Regal
  - Gut beschriften!
- Magnetbandroboter (Silo)
  - Roboter bedient Magnetbänder (Kassetten)
  - 10x schneller als Mensch
- CD / DVD - Juke-Boxes
  - Roboterarm extrahiert jeweiliges Medium
  - Hohe Lebensdauer (30 Jahre)
    - Wahrscheinlicher, dass kein Lesegerät mehr existiert

Vergleich (Stand 2001)

15

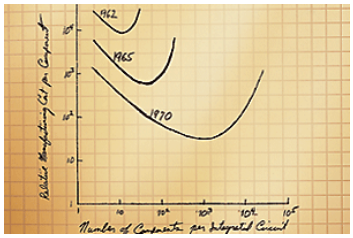


Moore's Law

16

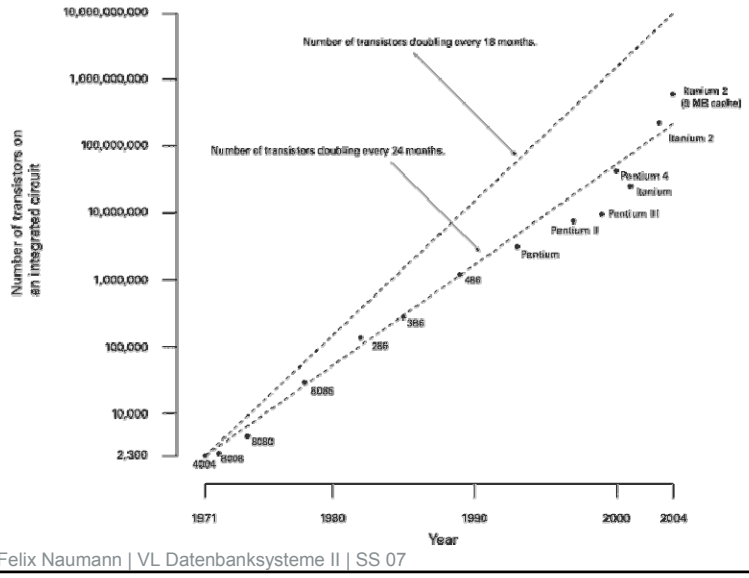
Exponentielles Wachstum vieler Parameter

- Verdopplung alle 18 Monate
  - Prozessorgeschwindigkeit
  - Hauptspeicherkosten pro Bit
  - Anzahl Bits pro cm^2
  - Diskkosten pro Bit
  - Kapazität der größten Disks



- Aber: Sehr langsames Wachstum
  - Zugriffsgeschwindigkeit im Hauptspeicher
  - Rotationsgeschwindigkeit von Disks
- Latenz wächst: Bewegung von Daten scheint immer langsamer (im Vergleich zum Prozessor)

Moore's Law



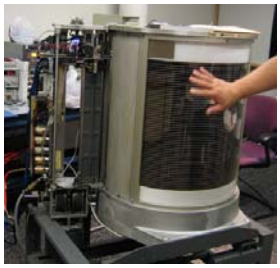
Übersicht

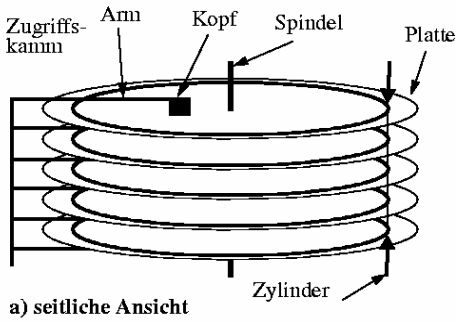
- Speicherhierarchie
- Disks
- Effiziente Diskoperationen
- Zugriffsbeschleunigung
- Diskausfälle
- RAID 0 – 6

Aufbau

19

- Mehrere (5-10) gleichförmig rotierende Platten
- Für jede Plattenoberfläche (10-20) ein Schreib-/Lese-Kopf
  - Gleichförmige Bewegung
- Plattenoberfläche ist in Spuren eingeteilt
- Spuren sind als Sektoren fester Größe formatiert
  - Anzahl Sektoren pro Spur kann sich unterscheiden
- Übereinandergeordnete Spuren sind ein Zylinder



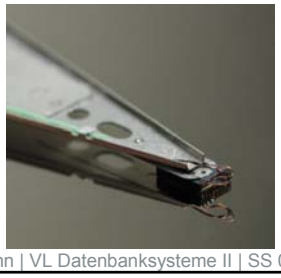


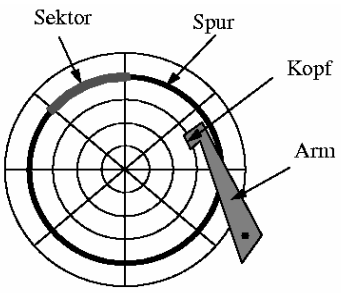
a) seitliche Ansicht

Aufbau

20

- Sektoren (1-8 KB) kleinste **physische** Leseeinheit
  - Größe ist von Hersteller festgelegt
  - Mehr Sektoren auf äußeren Spuren
- Lücken zwischen Sektoren nehmen ca. 10% der Fläche ein.
  - Zum Auffinden der Sektoranfänge
- Blöcke sind die **logische** Übertragungseinheit
  - Können aus einem oder mehreren Sektoren bestehen





Disk Controller

- Kontrolliert eine oder mehrere Disks
- Kontrolliert Bewegung der Schreib-/Lese-Köpfe
- Wählt Plattenoberfläche und Sektor, von der gelesen wird
  - Kontrolliert Start und Ende eines Sektors
- Überträgt Bits zwischen Disk und Hauptspeicher

Disk Eigenschaften

Eigenschaft	2004	1998	1990	1985	1970
Mittlere Zugriffs-bewegungszeit (ms)	5	8	12	16	30
Umdrehungszeit (ms)	6	6	14	16,7	16,7
Spurkapazität (KB)	420	100	56	47	13
Transferrate (MB/s)	30-40	15	4,2	3	0,8
Zylinder (#)	90.000	5.000	2226	2655	411
Kapazität (GB)	150	10	1,89	1,89 (?)	0,094

Datenbankgrößen

23

- 0,5 KB = Buchseite
- 1KB = 1000 Byte
- 30KB = gescannte, komprimierte Buchseite
- 1MB = 1000000 Byte
- 5MB = Bibel als Text
- 20MB = Gescanntes Buch
- 500MB = CD-ROM; Oxford English Dictionary
- 1GB = 1000000000 Byte
- 4,7 GB = DVD
- 10 GB = komprimierter Spielfilm
- 100GB = ein Stockwerk einer Bibliothek
- 200GB = Kapazität eines VHS Bandes
- 1TB = 1000000000000 Byte
- 1TB = Bibliothek mit 1 Mill. Bände
- 20TB = Library of Congress als Textdatei
- 1PB = 1000000000000000 Byte
- 1PB = eingescannte Bände einer Nationalbibliothek
- 15PB = weltweite Plattenproduktion 1996
- 200 PB = weltweite Magnetbandproduktion in 1996

Beispiel – Megatron 747 disk

24

- Eigenschaften
  - 8 Platten mit 16 Plattenoberflächen
  - 2^{14} = 16384 Spuren pro Oberfläche
  - Durchschnittlich 2^{7} = 128 Sektoren pro Spur
  - 2^{12} = 4096 Byte pro Sektor
- Gesamtkapazität?
  - 16 x 16384 x 128 x 4096 = 2^{37} Byte = 128 GB
- Blocks z.B. 2^{14} Byte
  - => 4 Sektoren pro Block
  - => 32 Blöcke pro Spur
- Bitdichte (3,5" Laufwerk)
  - Pro Spur 2^{7} x 2^{12} Byte = 2^{19} = 512KB = 4MBit
  - Spurlänge: $3{,}5\pi \approx 11$"
  - 10% Lücken => 9,9"
  - => 420.000 MBit / inch

Disk-Zugriffseigenschaften

- Voraussetzungen für Zugriff auf einen Block (lesend oder schreibend)
  - Kopf ist bei Zylinder positioniert, der die Spur mit dem Block enthält.
  - Disk rotiert so, dass Sektoren mit dem Block unter den Kopf gelangen.
- Latenzzeit
  - Zeit zwischen Anweisung, einen Block zu lesen, bis zum Eintreffen des Blocks im Hauptspeicher

Latenzzeit

Latenzzeit ist Summe aus vier Komponenten:

1. Kommunikationszeit zwischen Prozessor und Disk Controller
   - Bruchteil einer Millisekunde -> ignorieren
   - Annahme hier: Keine Konkurrenz
2. Seektime (Suchzeit) zur Positionierung des Kopfes
   - Zwischen 0 und 42ms
   - Startzeit (1ms), Bewegungszeit (10-40ms), Stopzeit (1ms)
3. Rotationslatenzzeit zur Drehung der Disk
   - Durchschnittlich ½ Umdrehung (5ms)
   - Optimierung durch Spur-Cache im Disk-Controller
4. Transferzeit zur Drehung der Disk bis alle Sektoren und die Lücken passiert sind
   - Ca. 16KB in ½ ms

Schreiben und Ändern

27

- Schreiben
  - Analog zum Lesen
  - Zur Prüfung muss eine Rotation gewartet werden
    - Checksums (später)
- Ändern
  - Nicht direkt möglich
  1. Lesen
  2. Im Hauptspeicher ändern,
  3. Schreiben
     - Mit Glück ist Kopf noch in der Nähe.
  4. Schreiboperation prüfen

Beispiel – Megatron 747 Disk

28

Wie lange dauert es, einen Block (16,284 Byte) zu lesen?

- Umdrehungsgeschwindigkeit 7200 U/min
  - => 1 Umdrehung in 8,33 ms
- Seektime
  - Start und Stopp zusammen 1ms
  - 1ms pro 1000 Zylinder, die überbrückt werden
    - Minimum (1 Zylinder): 1,001 ms
    - Maximum (16,383 Zylinder): 17,38 ms
- Minimale Zeit, den Block zu lesen?
  - Kopf steht schon richtig. Platte ist schon richtig rotiert
  - 4 Blöcke + 3 Lücken
  - 128 Lücken und 128 Sektoren pro Spur
  - Lücken bedecken 36°, Sektoren bedecken 324° des Kreises (360°)
  - 36 x 3/128 + 324 x 4/128 = 10,97°
  - (10,97/360) * 8,33ms = 0,253ms
- Maximale Zeit: selber
- Durchschnittliche Zeit: selber

Übersicht

- Speicherhierarchie
- Disks
- Effiziente Diskoperationen
- Zugriffsbeschleunigung
- Diskausfälle
- RAID 0 – 6

Algorithmen vs. DBMS

- Annahme bei Algorithmen
  - RAM-Modell für Berechnung: Daten passen in Hauptspeicher.
- Annahme bei Implementierung von DBMS
  - I/O-Modell: Daten passen **nicht** in Hauptspeicher.
- Geeignete Algorithmen funktionieren oft anders
  - Idee: I/O vermeiden
- Gleiches kann auch für Algorithmen im Hauptspeicher gelten.
  - Cache-Größe berücksichtigen!

I/O-Modell

31

- Beispiel: Einfache DBMS
  - Zu groß für Hauptspeicher
  - Eine Disk, ein Prozessor, viele Nutzer/Anfragen
- Stets eine Warteschlange an Zugriffsaufforderungen
  - First-come-first-served
  - Jede Aufforderung erscheint zufällig
    - Kopf ist also an zufälliger Position

Dominanz der I/O-Kosten

- Kosten der Bewegung eines Block zwischen Disk und Hauptspeicher sind wesentlich größer als Operationen auf den Daten im Hauptspeicher.
  => Anzahl der Blockzugriffe (lesend und schreibend) sind gute Approximation der Gesamtkosten und sollten minimiert werden.

Beispiel für das I/O-Modell: Indizes

32

- Relation R
- Anfrage sucht nach Tupel mit Schlüssel *k*.
- Index auf Schlüsselattribut
  - Variante A des Index sagt, auf welchem Block das Tupel liegt.
  - Variante B sagt zusätzlich, wo auf dem Block das Tupel liegt.
- Welche Index ist geeigneter?

- 11 ms um 16KB Block zu lesen
  - Entspricht viele Millionen Prozessoranweisungen
- Suche nach *k* auf dem Block kostet höchstens Tausende Prozessoranweisungen.
  - Aber: Zusätzliche Informationen in B nehmen Platz ein.

Beispiel für das I/O-Modell: Sortierung

33

- Relation R
  - 10 Million Tupel
  - Verschiedene Attribute, eines ist Sortierschlüssel
    - Nicht unbedingt Primärschlüssel
    - Vereinfachende Annahme: Sortierschlüssel ist UNIQUE
  - Gespeichert auf Diskblöcken der Größe 16.384 BYTE (2^{14})
  - 100 Tupel passen auf einen Block
    => Tupelgröße ca. 160 Byte
    => R hat 100.000 Blöcke
- Disk Megatron 747
- Hauptspeicherpuffer 100 MB (= 100 x 2^{20})
  - => 100 x 2^{20} / 2^{14} = 6400 Blöcke im Hauptspeicher
- Sortierung soll Anzahl der Lese- und Schreib-Operationen minimieren.
  - Wenig „Durchläufe" durch die Daten

Merge Sort

34

- Idee: Mische zwei (oder mehr) sortierte Listen zu einer größeren sortierten Liste.
  - Wähle aus den sortierten Listen stets das kleinste Element und füge es der großen Liste hinzu.

	Liste 1	Liste 2	Outputliste
1.	1,3,4,9	2,5,7,8	-
2.	3,4,9	2,5,7,8	1
3.	3,4,9	5,7,8	1,2
4.	4,9	5,7,8	1,2,3
5.	9	5,7,8	1,2,3,4
6.	9	7,8	1,2,3,4,5
7.	9	8	1,2,3,4,5,7
8.	9	-	1,2,3,4,5,7,8
9.	-	-	1,2,3,4,5,7,8,9

Merge Sort

- Aufwand
  - O(n+m)
  - Rekursiv auf immer größeren Listen: log n Durchläufe
  - Zusammen O(n log n)
- Rekursion
  - Teile eine Liste mit mehr als einem Element beliebig in zwei gleich lange Listen auf.
  - Sortiere rekursiv die kleineren Listen.
  - Merge sie zusammen.

Two-Phase, Multiway Merge-Sort (TPMMS)

- TPMMS wird in vielen DBMS eingesetzt.
- Phase 1
  - Sortiere Teilstücke, die in Hauptspeicher passen.
  - => Viele sortierte Teillisten
- Phase 2
  - Mische sortierte Teilliste in einzige große Liste.

TPMMS – Phase 1

- Rekursionsanfang nicht nur mit 1 oder 2 Elementen!
- Sortierung der Teillisten z.B. mit Quicksort

1. Fülle verfügbaren Hauptspeicher mit Diskblöcken aus Originalrelation
2. Sortiere im Hauptspeicher
3. Schreibe sortierte Tupel auf neue Blöcke der Disk

- Beispiel
  - 6400 Blöcke in Hauptspeicher; 100.000 Blöcke gesamt
  - => 16 Füllungen des Hauptspeichers
    - Letzte ist kleiner
  - Aufwand: 200.000 I/Os
    - 100.000 Blöcke lesen
    - 100.000 Blöcke schreiben
  - Zeit: 11ms pro I/O => 2200 sec = 37 min.
    - => Prozessorzeit vernachlässigbar

TPMMS – Phase 2

- Erste Idee: Paarweises merging
  - 2 log n mal lesen und schreiben
  - Im Beispiel: Ein Durchlauf für 8 sortierte Teillisten, einer für 4, einer für 2 und ein letzter für das Ergebnis
    - Insgesamt 8 I/Os jedes Blocks
- Bessere Idee: Ersten Block **jeder** Teilliste lesen

1. Suche kleinsten Schlüssel unter den ersten Tupel aller Blöcke.
   - Lineare Suche, Priority Queue
2. Bewege dieses Element in ein Output-Block.
3. Falls Output-Block voll ist schreibe auf Disk.
4. Falls ein Inputblock leer ist, lese nächsten Block aus gleicher Liste.

- Aufwand:
  - 2 I/Os pro Block (37 min)
- TPMMS insgesamt: 74 min

Blockgröße

39

- Beobachtung
  - Je größer Blockgröße, desto weniger I/Os
  - (Transferzeit erhöht sich etwas)
- Beispiel bisher
  - Blockgröße 16KB
  - Latenzzeit 10,63ms
- Beispiel neu
  - Blockgröße 512KB
  - Latenzzeit 20ms
  - Nur 12.500 I/Os für Sortierung
  - 17x Beschleunigung!
- Nachteile
  - Blocks sollten sich nicht über mehrere Spuren erstrecken.
  - Kleine Relationen nutzen nur Bruchteile eines Blocks.
  - Viele Datenstrukturen bevorzugen viele kleine Elemente anstelle von wenigen großen.

TPMMS – Grenzen

40

- Notation: Blockgröße B, Hauptspeicher M, Tupelgröße R
- M/B Blöcke passen in Hauptspeicher
- 1 Outputblock
- Phase 1 kann also maximal M/B -1 sortierte Teillisten herstellen.
- Ebenso oft kann man also den Hauptspeicher mit Tupeln füllen und sortieren
  - M / R Tupel
- Zusammen: (M/R) * ((M/B) -1) ≈ M^2/RB
- Beispiel
  - M = 104.857.600, B = 16,384, R = 160
  - Zusammen: 4,2 Billionen Tupel (ca. 2/3 Terabyte)
- Lösung: Dritte Phase
  - M^3 / RB^2 Tupel (ca. 4,3 Petabyte)

Übersicht

- Speicherhierarchie
- Disks
- Effiziente Diskoperationen
- Zugriffsbeschleunigung
- Diskausfälle
- RAID 0 – 6

Zugriffsbeschleunigung – Ideen

- Annahmen bisher
  - Nur eine Disk
  - Zugriffe zufällig (viele kleine Anfragen)
- Verschiedene Verbesserungsideen
  - Blöcke die gemeinsam gelesen werden, auf gleichen **Zylinder** platzieren.
  - Daten auf mehrere (kleine) Disks **verteilen**.
    - Umgeht die langsamen Seek-times
  - **Spiegelung** von Daten auf mehrere Disks.
  - Verwendung eines Disk-**Scheduling** Algorithmus.
  - **Prefetching**

Daten gemäß Zylinder organisieren

43

- Seektime verbraucht ca. 50% der I/O Zeit
- Idee: Daten, die zusammen gelesen werden, auf gleichen Zylinder platzieren
  - Z.B. Relationen
  - Zur Not: Mehrere nebeneinander liegende Zylinder
- Im besten Fall: Nur einmal Seektime
  - Maximale Zugriffszeit der Disk wird erreicht.

Zylinderorganisation – Beispiel

44

- Megatron 747
  - Transferzeit pro Block: ¼ ms
  - Seektime 6,46 ms
  - Rotationslatenzzeit 4,17 ms
  - 16.384 Zylinder á 512 Blocks
- Sortierung von 10.000.000 Tupeln dauerte 74 min.
  - 100.000 Blöcke auf 196 Zylinder
- Phase 1 – Lesen
  - Hauptspeicher (6.400 Blocks) wird 16 mal gefüllt
  - Jeweils aus 13 Zylindern (196 / 16)
    - Zylinder liegen nebeneinander: 1ms um Spur zu wechseln
  - Jeweils: 6,46 ms + 12 ms + 1,6 sec = 1,6 sec
    1 Seek Spurwechsel Transfer 6400 Blöcke
  - 1,6 sec x 16 = 26 sec (<< 18 min!)
- Phase 1 – Schreiben: Analog – zusammen 52 sec
- Phase 2 wird nicht beschleunigt
  - Lesen aus verschiedenen (verteilten) Teillisten
  - Schreiben zwar sequentiell, aber unterbrochen von Leseoperationen

Mehrere Disks

45

- Problem: Köpfe einer Disk bewegen sich stets gemeinsam.
- Lösung: Mehrere Disks mit unabhängigen Köpfen
  - Annahme: Controller, Hauptspeicher, Bus kommen mit höheren Transferraten klar.
  - Wirkung: Division aller Zugriffszeiten durch Anzahl Disks!
- Megatron 737 wie 747, aber nur 4 Plattenoberflächen
  - 4 x 737 = 747
- TPMMS Phase 1
  - Lesen: Von jeder Platte nur ¼ der Daten => 400ms
    - Wg. Freiheit, Teillisten selbst zu definieren
  - Schreiben: Jede Teilliste wird auf 4 Disks verteilt
    - Wie zuvor 13 Zylinder pro Disk (weniger Oberflächen)
  - Zusammen nur 13 sec (statt 52 sec; statt 37 min)

Mehrere Disks

46

- TPMMS Phase 2
  - Verteilung nützt zunächst nichts.
- Trick für Lesen: Man kann Tupel verarbeiten bevor Block vollständig im Hauptspeicher gelandet ist
  - So können potenziell mehrere Blöcke parallel (jeweils einer pro Teilliste) geladen werden.
  - Vorsicht: Sehr delikate Implementierung
  - Frage: Warum klappt das nicht immer?
- Schreiben des Outputs
  - Mehrere Output Blöcke
  - Einer wird gefüllt während die anderen drei geschrieben werden.
- Geschätzte Beschleunigung: Faktor 2-3
  - Immerhin!

Spiegelung

- Idee: Zwei oder mehr Disks halten identische Kopien
  - Mehr Sicherheit vor Datenverlust
  - Beschleunigter Zugriff
- TPMMS, Phase 2, lesen: Trick bei mehreren Disks klappt nicht immer
  - Falls Blöcke für verschiedene Teillisten auf gleicher Disk liegen
  - Bei Spiegelung kann garantiert werden, dass immer so viele Blöcke parallel gefüllt werden wie Spiegelungen vorhanden sind.
- Weiterer Vorteil, auch ohne Parallelität
  - Auswahl der Disk möglich (Kopf, der am dichtesten an relevanter Spur steht)
- Anmerkungen
  - Teuer
  - Keine Schreibvorteile

Disk Scheduling

Idee: Controller entscheidet welche Anweisungen zuerst ausgeführt werden.

- Nützlich bei vielen kleinen Prozessen, je auf wenigen Blöcken
  - OLTP
- Elevator Algorithmus
  - Fahrstuhl fährt in Gebäude hoch und runter
    - Hält an Stockwerken an wenn jemand ein oder aussteigen will.
    - Dreht um falls weiter oben/unten keiner mehr wartet.
  - Diskkopf streicht über Oberfläche einwärts und auswärts
    - Hält an Zylinder an, wenn es eine (oder mehr) Anweisung für einen Zugriff gibt.
    - Dreht um falls in jeweiliger Richtung keine Anweisungen mehr ausstehen.

Elevator Algorithmus

49

Kopf steht bei 2000

Angefragter Zylinder	Eintreffen der Anweisung	Zeitpunkt der Fertigstellung
2000	0	4,42
6000	0	13,84
14000	0	27,26
4000	10	42,68
16000	20	60.10
10000	30	71,52

Angefragter Zylinder	Zeitpunkt der Fertigstellung
2000	4,42
6000	13,84
14000	27,26
16000	34,68
10000	46,10
4000	57,52

Nur kleine Verbesserung, aber auch nur Mikro-Beispiel

Zahlen sind zum selber nachrechnen

Elevator Algorithmus

50

- Verbesserung steigt mit durchschnittlicher Anzahl von wartenden Anweisungen.
  - So viele wartende Anweisungen wie Anzahl Zylinder
    => Jeder Seek geht über nur wenige Zylinder.
    => Seek-time wird minimiert.
  - Mehr Anweisungen als Zylinder
    => Mehrere Anweisungen pro Zylinder
    => Sortierung um den Zylinder herum möglich
    => Rotationslatenz wird minimiert
- Nachteil?
  - Aber: Wartezeiten für einzelnen Anweisungen werden sehr hoch!

Prefetching

Idee: Wenn man voraussagen kann, welcher Blöcke gebraucht werden, kann man sie früh (bzw. während man sie sowieso passiert) in den Hauptspeicher laden.

- TPMMS, Phase 2, lesen: 16 Blöcke für die 16 Teillisten reserviert
  - Viel Hauptspeicher frei
  - Reserviere zwei Blöcke pro Teilliste
    - Fülle einen während der andere abgearbeitet wird.
    - Wenn einer entleert ist, wechsele zum anderen.
  - Aber: Zeit wird nicht verbessert

Idee: Kombination mit guter Zylinderorganisation

- TPMMS, Phase 1, schreiben: Schreibe Teillisten auf ganze, aufeinanderfolgende Zylinder
- TPMMS, Phase 2, lesen: Lese ganze Zylinder, wenn aus einer Liste ein neuer Block benötigt wird.

Idee für das Schreiben analog: Zögere Schreiboperationen hinaus bis ganzer Zylinder geschrieben werden kann.

Zusammenfassung

Zwei Arten von Anwendungen

1. Viele Blöcke werden in bekannter Folge gelesen oder geschrieben. Nur ein Prozess. (TPMMS, Phase 1)
2. Viele kleine, parallele, unvorhersagbare Prozesse (ähnlich TPMMS, Phase 2)

Fünf Tricks

- Zylinderorganisation
- Mehrere Disks
- Spiegelung
- Scheduling mit Elevator Algorithmus
- Prefetching

Zusammenfassung

Zwei Arten von Anwendungen

1. Viele Blöcke werden in bekannter Folge gelesen oder geschrieben. Nur ein Prozess. (TPMMS, Phase 1)
2. Viele kleine, parallele, unvorhersagbare Prozesse (ähnlich TPMMS, Phase 2)

Zylinderorganisation

- Idee: Daten, die zusammen gelesen werden, auf gleichen Zylinder platzieren.
- Vorteil: Perfekt für 1.
- Nachteil: Hilft nicht für 2.

Zusammenfassung

Zwei Arten von Anwendungen

1. Viele Blöcke werden in bekannter Folge gelesen oder geschrieben. Nur ein Prozess. (TPMMS, Phase 1)
2. Viele kleine, parallele, unvorhersagbare Prozesse (ähnlich TPMMS, Phase 2)

Mehrere Disks (RAID 0)

- Idee: Mehrere Disks mit unabhängigen Köpfen beheben das Problem der gemeinsamen Kopfbewegung
- Vorteil: Erhöht Schreib/Lese Rate für 1. und 2.
- Problem: Operationen auf gleicher Disk werden nicht beschleunigt. Gesamtbeschleunigung nicht n-fach
- Nachteil: Teurer bei gleicher Diskkapazität

Zusammenfassung

55

Zwei Arten von Anwendungen

1. Viele Blöcke werden in bekannter Folge gelesen oder geschrieben. Nur ein Prozess. (TPMMS, Phase 1)
2. Viele kleine, parallele, unvorhersagbare Prozesse (ähnlich TPMMS, Phase 2)

Spiegelung (RAID 1)

- Idee: Identische Kopien auf mehreren Disks
- Vorteile: Erhöht Schreib/Lese Rate für 1. und 2.
  Beschleunigung ist n-fach
  Erhöhte Fehlertoleranz
- Nachteil: Kosten verdoppelt bzw. ver-n-facht

Zusammenfassung

56

Zwei Arten von Anwendungen

1. Viele Blöcke werden in bekannter Folge gelesen oder geschrieben. Nur ein Prozess. (TPMMS, Phase 1)
2. Viele kleine, parallele, unvorhersagbare Prozesse (ähnlich TPMMS, Phase 2)

Scheduling und Elevator Algorithmus

- Idee: Controller entscheidet welche Anweisungen zuerst ausgeführt werden.
- Vorteil: Reduziert durchschnittliche Lese/Schreib-Zeit für 2.
- Problem: Funktioniert am besten bei vielen wartenden Anweisungen. Durchschnittliche Wartezeit wird erhöht.

Zusammenfassung

Zwei Arten von Anwendungen

1. Viele Blöcke werden in bekannter Folge gelesen oder geschrieben. Nur ein Prozess. (TPMMS, Phase 1)
2. Viele kleine, parallele, unvorhersagbare Prozesse (ähnlich TPMMS, Phase 2)

Prefetching

- Idee: Künftig benötigte Blöcke frühzeitig laden
- Vorteil: Beschleunigt 2.; es ist bekannt, welche Daten benötigt werden, es ist nur unklar wann.
- Nachteil: Benötigt zusätzlichen Hauptspeicher

Übersicht

- Speicherhierarchie
- Disks
- Effiziente Diskoperationen
- Zugriffsbeschleunigung
- Diskausfälle
- RAID 0 – 6

Fehlerklassifikation

1. Transaktionsfehler
   - Führt zu Transaktionsabbruch
   - Fehler in der Anwendung (division by zero)
   - **`abort`** Befehl
   - Abbruch durch DBMS (Deadlock)
2. Systemfehler
   - Absturz in DBMS, Betriebssystem, Hardware
   - Daten im Hauptspeicher werden zerstört
3. Medienfehler
   - Head-crash, Controller-Fehler, Katastrophe
   - Daten auf Disk werden zerstört
   - Jetzt mehr…

Diskfehlerarten (Medienfehler)

1. Sporadischer Fehler
   - Erfolgloser Lese- oder Schreibversuch
   - Wiederholte Versuche führen zu Erfolg
2. Medienverfall
   - Korrupte Bits auf Disk
   - Wiederholte Leseversuche führen nicht zum Erfolg
3. Schreibfehler
   - Schreibanweisung kann nicht ausgeführt werden
   - Z.B. Stromausfall
4. Diskausfall
   - Gesamte Disk kann nicht gelesen werden
   - Plötzlich und permanent

Jetzt: Checksums Stable Storage

Gleich: RAID

Sporadische Fehler

61

- Idee: Redundante Bits speichern Status-Informationen
  - „Correct" / „incorrect"
- Leseanweisung gibt Paar *(w, s)* zurück
  - *w*: Die Daten
  - *s*: Der Status
- Falls *s* „incorrect" ist: bis zu 100 Wiederholungsversuche
- *s* kann selbst Fehlerhaft sein
  - Beliebig genau durch weitere Redundanz
- Nach Schreibanweisung analog
  - *s* lesend prüfen

Checksums / Parity Bit

62

Idee: Jeder Sektor erhält einige Bits zur Speicherung einer Checksum

- Checksum ist abhängig von Daten im Sektor
- Problem: Checksum könnte zufällig zu inkorrekten Daten passen
  - Mehr Bits verringern diese Wahrscheinlichkeit
- Parity Bit als Checksum
  - Ungerade Anzahl 1en => Parity Bit ist 1
  - Gerade Anzahl 1en => Parity Bit ist 0
- => Anzahl 1en im gesamten Sektor ist immer gerade!
- Beispiel: 01101000 + parity bit 1 => 011010001
- Beispiel: 11101110 + parity bit 0 => 111011100
- Jeder 1-Bit Fehler kann leicht erkannt werden
  - Fehler übersehen: 50% bei mehr als einem 1-Bit Fehler
- Idee: 8 parity Bits – eines für jedes Bit der Bytes
  - Wkt. Fehler zu übersehen: 1/256
- Allgemein: n parity Bits => Wahrscheinlichkeit $1/2^n$

Stable Storage

63

- Problem: Checksums helfen nicht, Fehler zu **beheben**.
- Problem: Bei fehlerhaftem Schreiben überschreibt man auch alte Daten.
  - Beispiel: Kontostandveränderung
- Idee: Sektoren werden zu Paaren zusammengeschlossen.
  - Linke Kopie X_L
  - Rechte Kopie X_R
  - Annahme: Durch Checksums können Lese- und Schreibfehler **erkannt** werden.

Stable Storage Protokolle

64

- Schreibprotokoll
  1. Schreibe Wert auf X_L.
  2. Prüfe mittels Checksum auf Korrektheit.
  3. Falls inkorrekt: Wiederhole 1. und 2. *n*-mal
  4. Falls immer noch inkorrekt: Medienfehler; neuen Sektor allozieren und wiederholen
  5. Wiederhole 1. – 4. auf X_R.
- Leseprotokoll
  1. Lese Wert von X_L.
  2. Falls inkorrekt, wiederhole 1. *n*-mal
  3. Falls immer noch inkorrekt, wiederhole 1. und 2. mit X_R.

Stable Storage – Medienverfall

- Daten können immer vom jeweils anderen Sektor gelesen werden.
- Falls X_R defekt
  - Leseprotokoll merkt nichts
  - Schreibprotokoll wird Fehler erkennen
- Falls X_L defekt
  - Leseprotokoll springt zu XR
  - Schreibprotokoll bemerkt Fehler ebenfalls
- Beide defekt: Unwahrscheinlich

Stable Storage – Schreibfehler

- Stromausfall während Schreiboperation
  - Z.B Kontostandsveränderung
  - Wert im Hauptspeicher ist verloren
  - Wert ist erst zur Hälfte geschrieben (inkorrekt)
- Alter Wert kann immer gelesen werden
  - Fehler beim Schreiben auf X_L
    - Status auf X_L ist inkorrekt
    - Status auf X_R ist korrekt
      - » Kann verwendet werden, um X_L zu reparieren
  - Fehler nach Schreiben auf X_L
    - X_R sollte mit Wert von X_L repariert werden.

Übersicht

- Speicherhierarchie
- Disks
- Effiziente Diskoperationen
- Zugriffsbeschleunigung
- Diskausfälle
- RAID 0 – 6

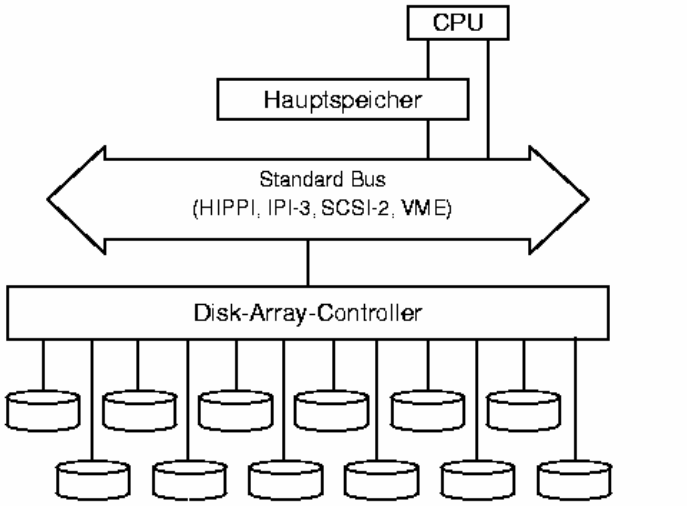
Diskausfälle

Generell: **Redundanz** zur Vermeidung von Datenverlust

- RAID: Redundant Arrays of Independent Disks
- RAID: Redundant Arrays of Inexpensive Disks
- „Commodity Hardware" mit Software zu schnellen und fehlertoleranten Systemen zusammenbauen.

Disk Arrays ➔ RAID-Systeme

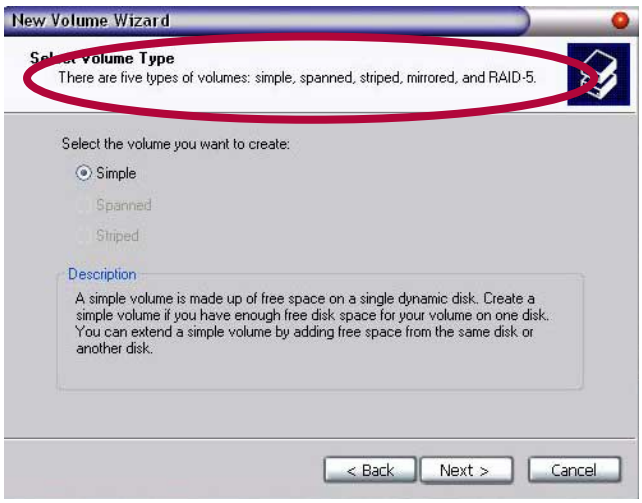
69



Quelle: Folien Ulf Leser (HU)

Soft RAID in Windows XP

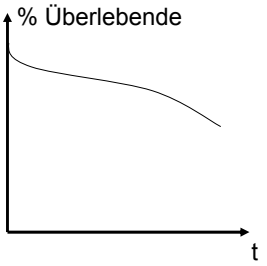
70



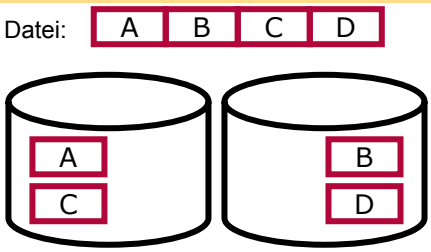
Quelle: Folien Ulf Leser (HU)

Fehlertoleranz

- Diskarray mit N Disks
  - Mean-time-to-failure (MTTF)
    - Ca. 10 Jahre
  - Mean-time-to-repair (MTTR)
  - Mean-time-to-data-loss (MTTDL)
  - Ohne Fehlertoleranzmechanismen **erhöhte** Ausfallwahrscheinlichkeit
    - MTTDL = MTTF / N



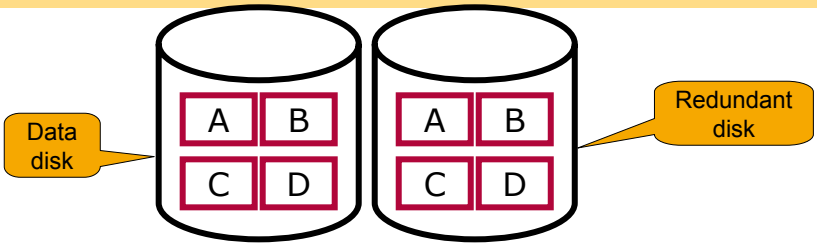
RAID 0 – Striping



- Entspricht „Mehrere Disks" (s.o.)
- Doppelte Bandbreite beim sequentiellen Lesen der Datei
  - Zyklische Verteilung der Blöcke, wenn alle Blöcke mit gleicher Häufigkeit gelesen/geschrieben werden (*round robin*)
  - Alternative: Verteilung nach Zugriffshäufigkeit (Optimierungsproblem)
    - Zugriffshäufigkeit aber i.d.R. nicht bekannt
- Aber: Keine Beschleunigung für Lesen eines einzelnen Blocks
- Aber: Dateiverlust wird immer wahrscheinlicher, je mehr Disks man verwendet.

Quelle: Folien Ulf Leser (HU)

RAID 1 – Mirroring



- Wie „Mirroring" (s.o.)
- Datensicherheit durch Redundanz aller Daten
- Aber: Doppelter Speicherbedarf
- Lastbalancierung beim Lesen (wie RAID 0)
- Außerdem „konkurrierendes" Lesen einer 1-Block Datei möglich.
  - Der schnellere Kopf gewinnt.
- Beim Schreiben müssen beide Kopien geschrieben werden.
  - Kann parallel geschehen

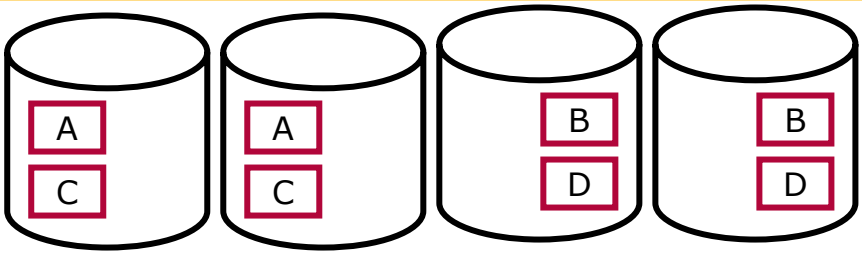
Quelle: Folien Ulf Leser (HU)

RAID 1 – Datenverlust (Beispiel)

- 2 Disks mit MTTF = 10y
  - Wkt 10%, dass Disk in gegebenem Jahr ausfällt.
- Bei Ausfall einer Disk
  - Disk ersetzen und Dateien von redundanter Disk kopieren.
  - Mögliches Problem: Redundante Disk fällt während dieses Kopiervorgangs aus.
- Wahrscheinlichkeit für Ausfall?
  - 3h für Kopieren = 1/8 Tag = 1/2920 Jahr
  - Wkt. eines Ausfalls in einem Jahr = 1/10
  - Wkt. dass Disk während des Kopierens ausfällt = 1/29.200
  - Eine der beiden Disks fällt alle 5 Jahre aus.
  - MTTDL mit RAID 1: 5 x 29.200 = 146.000 Jahre
- Annahme?
  - Unabhängigkeit der Diskfehler
  - Stimmt diese Annahme?

RAID 0 + 1 – Mirrored Striping

75

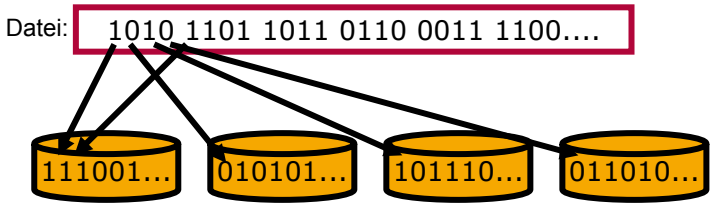


- Kombiniert RAID 0 und RAID 1
- Doppelter Speicherbedarf
- Erhöhte Ausfallsicherheit (durch RAID 1)
- Weiter erhöhte Lesegeschwindigkeit für Dateien
  - Datei kann von vier Disks parallel gelesen werden.
- Lesen eines einzelnen Blocks kaum schneller als ohne RAID
- Schreibgeschwindigkeit einer Datei verdoppelt sich.

Quelle: Folien Ulf Leser (HU)

RAID 2 - Striping auf Bit-Ebene

76

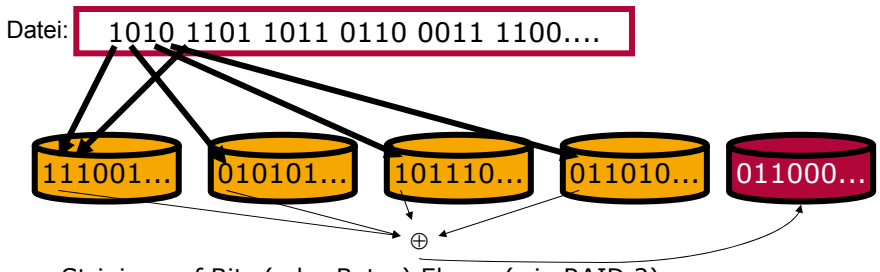


- Striping auf Bit- (oder Byte-) statt Blockebene
- Idealerweise höherer Durchsatz schon beim Lesen einzelner Blöcke
  - Aber: Lesen eines Sektors dauert i.d.R. genauso lange wie Lesen 1/8 Sektors
  - Also muss Aufteilung von Blöcken auf Sektoren beachtet werden
    - Typisch: 8-16 KB Blöcke, 512 Byte Sektoren = 16-32 Sektoren pro DB Block
- Keine Beschleunigung für mehrere parallele Leseoperationen
  - Jedes Lesen/Schreiben braucht alle Disks
- Verschlechterte Ausfallsicherheit

Quelle: Folien Ulf Leser (HU)

RAID 3 – Striping mit Parity Blocks

77

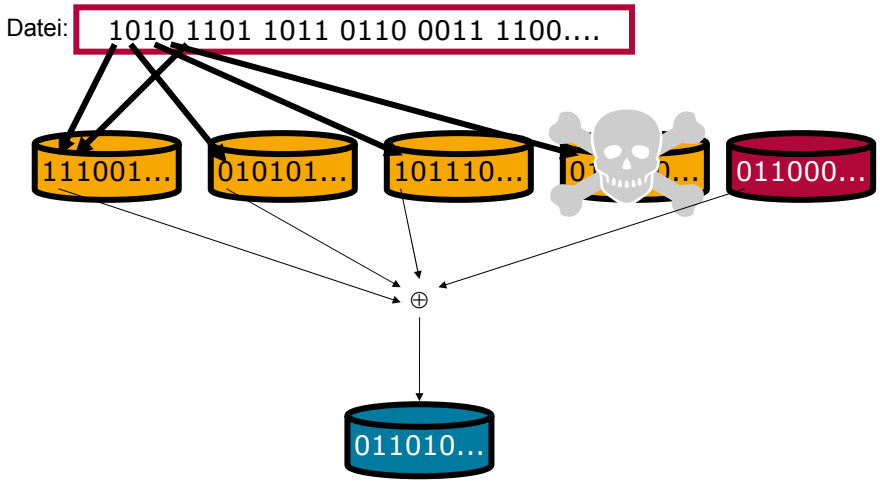


- Striping auf Bit- (oder Byte-) Ebene (wie RAID 2)
- Zusätzliche Disk für Parität der anderen Disks
  - Parität = bit-weises xor ⊕
  - Ausfall einer Disk kann kompensiert werden.
- Lesen/Schreiben eines Blocks erfordert Zugriff auf alle Disks.
- Deutlich weniger Platzbedarf als RAID 1, trotzdem Ausfallsicherheit

Quelle: Folien Ulf Leser (HU)

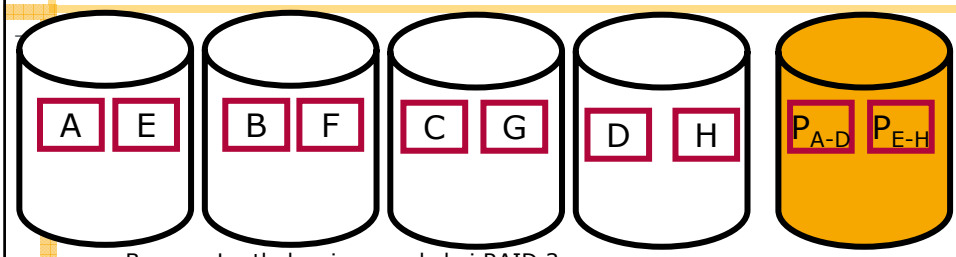
RAID 3 - Diskausfall

78



Quelle: Folien Ulf Leser (HU)

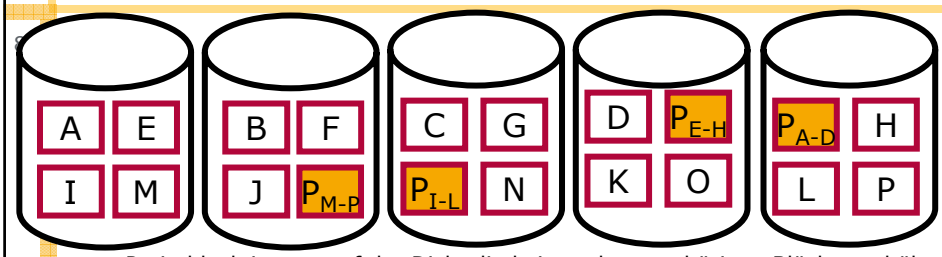
RAID 4 – Block Striping mit Parity



- Bessere Lastbalancierung als bei RAID 3
  - Verschiedene Prozesse können parallel von verschiedenen Disks lesen.
- Damit: Paritätsdisk ist Flaschenhals
  - Bei jedem Schreiben muss darauf zugegriffen werden.
  - Bei jedem Lesen muss u.U. darauf zugegriffen werden (zur Kontrolle).
  - Bei Modifikation von Block A zu A‘: $P'_{A-D} := P_{A-D} \oplus A \oplus A'$
- D.h. bei einer Änderung von Block A muss der alte Zustand von A und der alte Paritätsblock gelesen werden und der neue Paritätsblock und der neue Block A‘ geschrieben werden.

Quelle: Folien Ulf Leser (HU)

RAID 5 – Block Striping mit verteilter Parity



- Parityblock immer auf der Disk, die keinen der zugehörigen Blöcke enthält
- Wesentlich bessere Lastbalancierung als bei RAID 4
  - Paritätsdisk kein Flaschenhals mehr
  - Schreiben erfordert *n*-1 Disks für die Daten und 1 Disk für Parity
- Wird in der Praxis häufig eingesetzt.
  - Typischerweise langsamer als RAID 0+1
  - Dafür deutlich Platz sparender
- Guter Ausgleich zwischen Platzbedarf und Leistungsfähigkeit

Quelle: Folien Ulf Leser (HU)

RAID 6 – Hamming Codes

- Wie RAID 5 aber mit mehr Kontrollinformationen
- Erlaubt Wiederherstellung auch nach mehreren Diskausfällen.
- Höchste RAID-Stufe

- Nicht hier
  - Siehe z.B. http://de.wikipedia.org/wiki/Hamming_Code

Zusammenfassung

	Block-Striping	Bit-Striping	Kopie	Parität	Parität, dedizierte Disk	Verteilte Parität	Erkennen mehrerer Fehler
RAID 0	X						
RAID 1			X				
RAID 0+1	X		X				
RAID 2		X					
RAID 3		X		X	X		
RAID 4	X			X	X		
RAID 5	X			X		X	
RAID 6	X			X			X

- RAID 1 und vor allem RAID 5 sind weit verbreitet
  - RAID1: Einfach, schnell zu realisieren, erhöhte Ausfallsicherheit und Geschwindigkeit, aber verdoppelter Platzbedarf (billig)
  - RAID5: Relativ komplex, erhöhte Ausfallsicherheit und Geschwindigkeit, nur geringfügig erhöhter Platzbedarf; benötigt mindestens 3 Disks

Rückblick

- Speicherhierarchie
- Disks
- Effiziente Diskoperationen
- Zugriffsbeschleunigung
- Diskausfälle
- RAID 0 – 6