Großer Beleg

zum Thema

Ermittlung von Festplatten-Echtzeiteigenschaften

Martin Pohlack

Technische Universität Dresden
Fakultät Informatik
Institut für Systemarchitektur
Professur Betriebssysteme

7. Mai 2002

Verantwortlicher Hochschullehrer:
Prof. Dr. Hermann Härtig

Betreuer:
Dipl.-Inf. Lars Reuther

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Im Rahmen des *„Dresden Real-Time Operating System"* (DROPS) wird an einem Echtzeitdateisystem gearbeitet, welches parallele Datenströme mit weichen und harten Echtzeitanforderungen unterstützt. Dieses Projekt beinhaltet u. a. ein zusagenfähiges SCSI-Subsystem [Meh98] und ein echtzeitfähiges Dateisystem [Reu98].

Um Echtzeitdienste anbieten zu können, muss bekannt sein, wie leistungsfähig die zugrunde liegenden Dienste und Geräte sind. Für Dateisysteme sind hier besonders Festplatten interessant.

Ziel dieser Belegarbeit ist es deshalb Werkzeuge zu entwickeln, die die benötigten Informationen über Festplatten möglichst automatisch ermitteln können. Diese Informationen lassen sich in zwei Teilbereiche gliedern, basierend auf der Art der Echtzeitanforderungen:

- Für die Zusicherung von *harten* Echtzeitanforderungen muss das Worst-Case-Verhalten der benutzten Platten bekannt sein.
- Für die Koordinierung von Anforderungen mit *weicher* Echtzeit muss die Verteilung der Bearbeitungszeiten für typische Zugriffsmuster bekannt sein.

Um bei harter Echtzeit die Einhaltung aller Zeitschranken garantieren zu können, muss mit dem Worst-Case geplant werden. Die Worst-Case-Bearbeitungszeiten einzelner Aufträge sind oft deutlich größer als die Durchschnittsbearbeitungszeiten, treten aber fast nie auf. Deswegen geht mit der Zusicherung von harten Echtzeitanforderungen meist eine schlechte Auslastung der benutzten Geräte einher.

Für viele Anwendungen sind harte Echtzeitanforderungen nicht notwendig. Weiche Echtzeit stellt hier einen guten Kompromiss dar. Die Auslastung der Geräte lässt sich deutlich steigern, wobei trotzdem einen festgelegter Anteil der Plattenaufträge rechtzeitig erfüllt wird.

Hamann et al. stellen in [HLR01] eine Kombination beider Verfahren vor. Anforderungen werden dabei in einen obligatorischen Anteil, der immer erfüllt werden muss, und einen optionalen Anteil, der zu einem gewissen Anteil zu erfüllen ist, zerlegt.

Für die Planung von weichen Echtzeitanforderungen ist die Verteilung der Bearbeitungszeiten wichtig. Die Verteilung lässt sich aus typischen Zugriffsmustern ermitteln.

Teilaufgaben, die die Werkzeuge die Ziel dieser Belegarbeit sind abdecken sollen, sind also:

- Ermittlung der Worst-Case-Bearbeitungszeiten für Plattenaufträge,
- Aufnahme von typischen Zugriffsmustern bestimmter Anwendungen,
- Abspielen dieser Zugriffsmuster auf verschiedenen Platten und

- Vermessung des Abspielvorgangs (z. B. Ermittlung der Verteilung der Bearbeitungszeiten) auf den unterschiedlichen Platten.

Bei Festplatten ist eindeutig ein Trend hin zu IDE-Platten festzustellen. Leistungsmäßig liegen sie fast gleich auf mit SCSI-Platten, da intern meist die gleichen Komponenten verbaut werden, finanziell sind sie deutlich attraktiver. Praktisch alle heute erhältlichen IDE-Platten können mit den gleichen Treibern angesteuert werden, während bei SCSI jeder Kontroller einen speziellen Treiber braucht. In der vorliegenden Arbeit werden kleine Modifikationen im Treiber für die genaue Zeitmessung benötigt. Eine Lösung für IDE-Platten ist deswegen breiter einsetzbar. Des Weiteren ist festzustellen, dass sich der Befehlsumfang neuerer IDE-Standards immer mehr dem von SCSI angleicht.

Diese Arbeit wird sich deshalb auf IDE-Platten beschränken.

Gliederung

Im folgenden Kapitel wird auf den prinzipiellen Aufbau von Festplatten und verwandte Arbeiten eingegangen. In Kapitel 3 wird der Entwurf der Werkzeuge, insbesondere der Modellansatz auf dem diese Arbeit aufbaut, genauer erläutert. Das darauf folgende Kapitel 4 beschreibt die Realisierung einzelner Aspekte der Werkzeuge und stellt Teilverfahren vor. Ein Überblick zum Stand der Implementierung und der gewonnenen Ergebnisse wird in Kapitel 5 gegeben. Den Abschluss bildet Kapitel 6 mit einer Zusammenfassung und Ansatzpunkten für weiterführende Arbeiten.

Anhang A enthält eine Zusammenstellung von praktischen Untersuchungsergebnissen, die mit den im Beleg entstandenen Werkzeugen gemessen wurden.

2 Stand der Technik

Im Aufbau von Festplatten liegt der Schlüssel zum Verständnis ihres zeitlichen Verhaltens. Er soll deswegen in diesem Kapitel genauer beleuchtet werden. Des Weiteren wird auf verwandte Arbeiten eingegangen, die teilweise als Grundlage für die vorliegende Arbeit dienten.

2.1 Aufbau von Festplatten

Das grundlegende Prinzip rotierende Scheiben als Speichermedium zu verwenden, wurde schon vor ca. 100 Jahren beim Grammophon benutzt. Ein statischer Abtastkopf überstreicht den Bereich zwischen äußerem und innerem Rand der Scheibe die als Informationsträger unter ihm entlang rotiert.

In den nächsten Abschnitten soll angefangen beim mechanischen Aufbau, über grundlegenden Vorgänge bis hin zur Organisation der Daten ein Bild vom Festplatten und ihrem Verhalten vermittelt werden. Abschließend wird das Zusammenspiel von Festplatte und Rechner bei einem Plattenzugriff gezeigt.

2.1.1 mechanischer Aufbau

Festplatten bestehen typischerweise aus einem Stapel von magnetisch beschichteten Platten, die auf einer gemeinsamen Achse gelagert sind. Beide Oberflächen der Scheiben können zur Datenspeicherung benutzt werden. Jede Oberfläche des Plattenstapels wird von einem eigenen Schreib-/Lesekopf bearbeitet. Diese Köpfe sind an einem Kamm angebracht und werden gemeinsam von einem Linearmotor zwischen äußerem und innerem Rand der beschichteten Oberflächen bewegt. Die einzelnen Oberflächen sind in Spuren aufgeteilt, welche in konzentrischen Kreisen um die Achse angeordnet sind. Jede Spur wird in eine Folge von Sektoren unterteilt, die gleiche Mengen an Nutzdaten aufnehmen können. Bei heutigen Festplatten sind das typischerweise 512 Byte. Bei älteren Modellen waren auch 256 und 1024 Byte üblich [RueWi94, Meh98]. Sektoren sind die kleinste Einheit, die eine Festplatte lesen oder schreiben kann.

Die Spuren, die auf den einzelnen Oberflächen den selben Radius haben, bilden zusammen einen Zylinder.

Der Plattenstapel wird mit einer konstanten Winkelgeschwindigkeit (CAV — constant angular velocity)[1] angetrieben. Festplatten im Desktop-Bereich erreichen heute 5.400 bzw. 7.200 u/min. Für Server werden Platten mit bis zu 10.000 evtl. sogar 15.000 u/min eingesetzt.

[1] CD-ROM Laufwerke z. B. werden zumindest teilweise mit konstanter linearer Geschwindigkeit (CLV — constant linear velocity) betrieben. Sie müssen deshalb, je nach Zugriffsposition auf der CD, die Drehzahl ständig anpassen, was sich in schlechteren Zugriffszeiten niederschlägt.

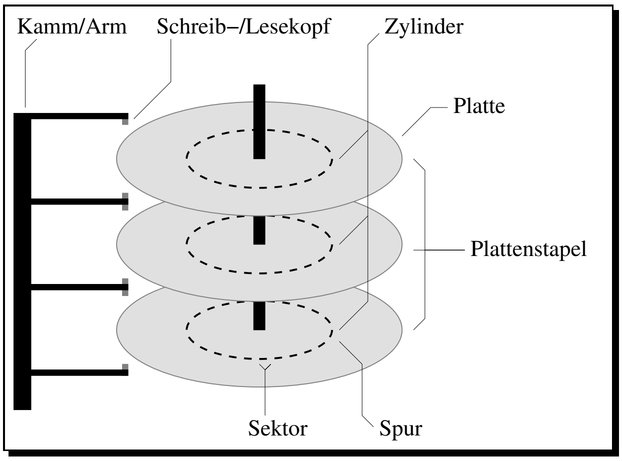


Abbildung 2.1: Aufbau einer Festplatte

2.1.2 Kammbewegung (seek)

Um die Köpfe von einem Zylinder auf einen anderen zugreifen zu lassen muss der Kamm sie dorthin bewegen. Dieser Vorgang wird als Seek bezeichnet. Ein typischer Seek läuft in vier Phasen ab:

1. **Beschleunigung**: Der Kamm wird in Richtung Zielzylinder beschleunigt, bis er seine Maximalgeschwindigkeit erreicht hat.
2. **Bewegung**: Der Kamm wird mit der maximalen Geschwindigkeit, die er in der Beschleunigungsphase erreicht hat weiter in Richtung Zielzylinder bewegt.
3. **Abbremsen**: Die Geschwindigkeit des Kammes wird abgebremst, so dass er ungefähr beim Zielzylinder zum Stehen kommt.
4. **Settle**: Der Kamm wird genau auf die Zielspur im Zylinder eingepasst.

Die Bewegungsphase wird nur bei längeren Seeks ausgeführt, ansonsten wird direkt von der Beschleunigungsphase aus abgebremst. Bei sehr kurzen Seeks oder beim Spurwechsel innerhalb eines Zylinders (Kopfumschaltung) kommt es nur zu einem Resettle. Der Kamm wird nur leicht an die neue Spur angepasst. Wird ein Schreibzugriff ausgeführt, so muss das Settle genauer ausgeführt werden. Bei Leseaufträgen kann eher auf das Medium zugegriffen werden, da durch fehlererkennenden Codes eine Falschpositionierung bemerkt wird. Im Gegensatz zu Schreibaufträgen können bei einem zu frühen Lesezugriff keine Daten auf der Platte zerstört werden. [RueWi94, IBM01]

Der Seek-Graph in Abbildung 2.2 zeigt auf dem rechten Zweig die Zeiten für Vorwärts-Seeks an. Ausgangspunkt für jeden Seek ist dabei immer Spur 0. Der linke Zweig zeigt Rückwärts-Seeks, wobei in der letzten Spur der Platte gestartet wird. Die Zeiten steigen monoton mit der Seek-Weite an. Aus der Symmetrie im Graph kann man auf ungefähr gleich schnelle Vorwärts- und Rückwärts-Seek-Vorgänge schließen. Ansätze die Seek-Funktion zu modellieren finden sich in [AAD97, RueWi94, Shr97, SMW98].

Für die Worst-Case-Betrachtungen dieser Arbeit ist eigentlich nur die Maximalzeit interessant.

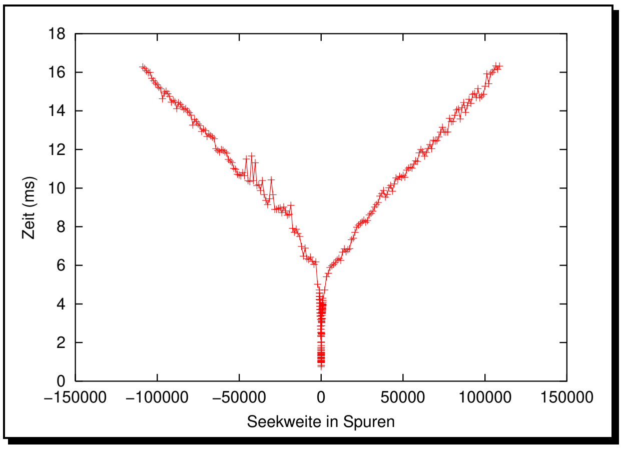


Abbildung 2.2: Seek-Graph für Vorwärts- und Rückwärts-Seeks

2.1.3 Versatz (skew)

Der erste Sektor einer Spur wird meist in einem deutlichen Winkelabstand zum letzten Sektor der vorherigen Spur angeordnet. Dieser Abstand wird als Versatz (*skew*) bezeichnet. Ziel dieser Anordnung ist die Beschleunigung des sequentiellen Plattenzugriffes.

Wird bei einem längeren Plattenauftrag über eine Spurgrenze hinweg gelesen oder geschrieben, so muss der Kamm bewegt bzw. der Kopf umgeschaltet werden, je nach dem ob eine Zylindergrenze überschritten wird oder nicht. Beide Vorgänge brauchen Zeit, einmal die Zylinderumschaltzeit (*cylinder switch time*), das andere mal die Kopfumschaltzeit (*head switch time*). Der Versatz sollte möglichst so eingestellt werden, dass direkt nachdem die maximale Zylinder- bzw. Kopfumschaltzeit verstrichen ist, der erste Sektor der neuen Spur unter dem Kopf ankommt. Wenn der Versatz zu groß ist, muss etwas gewartet werden, bevor der erste Sektor der neuen Spur erreicht ist. Ist er jedoch zu klein, muss fast eine ganze zusätzliche Umdrehung der Platte in Kauf genommen werden.

Beide Zeiten bewegen sich etwa im selben Bereich, da in beiden Fällen ein *Resettle* durchgeführt wird. Bei einer Kopfumschaltung ist meist auch eine leichte Korrektur des Kammes notwendig, da durch Fertigungstoleranzen die Spuren eines Zylinders nicht genau übereinander liegen.

2.1.4 Zonen

Um die Kapazität der Festplatte zu maximieren wird auf der gesamten Festplatte die gleiche hohe Datendichte angestrebt, die von der Beschichtung, der Genauigkeit der Schreib-/Leseköpfe und der Positioniergenauigkeit derselben begrenzt wird. Das dazu benutzte Verfahren heißt Zoned-Bit-Recording. Damit wird in den äußeren Bereichen der Festplatte eine höhere Sektorzahl pro Spur als in den inneren erreicht. Da die Drehzahl konstant ist, ergibt sich also in den äußeren Bereichen der Platte ein höhere Datenrate. Die Sektorzahl pro Spur kann von außen nach innen nicht in beliebigen Schritten verkleinert werden, u. a. deshalb, weil die Zahl der Sektoren pro Spur ganzzahlig bleiben muss. Bereiche gleicher Sektorzahl pro Spur werden zu Zonen zusammengefasst. Daher können Festplatten innerhalb einer Zone die selbe Datenrate liefern. Platten mit Zoned-Bit-Recording haben deswegen schnelle und langsame Zonen.

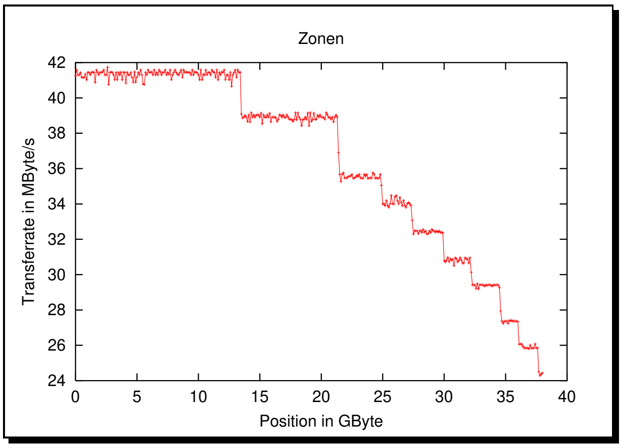


Abbildung 2.3: Zonen der Festplatte Seagate Barracuda ATA IV

Abbildung 2.3 wurde mit einer im Rahmen der vorliegenden Arbeit an *raw-devices* angepassten Version von `zcav [Coker]` aus der Benchmarksuite `bonnie++` erstellt. Dargestellt ist die Übertragungsrate der Festplatte bei lesendem Zugriff in Abhängigkeit von der Position auf der Platte. Man sieht sehr gut, dass die Datenrate mit größer werdenden logischen Adressen absinkt, die Festplatte also wahrscheinlich hohe logische Adressen in die inneren Zonen der Festplatte abbildet. Die erste Zone mit der höchsten Datenrate nimmt etwa 13,5 GByte dieser 40 GByte Platte ein.

2.1.5 Adressierung und Mapping

Festplatten stellen heute dem Rechner gegenüber eine kontinuierliche Folge von Speicherblöcken zur Verfügung. Einzelne Blöcke dieser Folge werden dabei vom Rechner aus über die logische Adresse angesprochen.

Die Umrechung auf physische Adressen erfolgte früher im Plattentreiber. Die physische Position auf Platte wurde dabei durch das CHS-Verfahren (cylinder-head-sector) beschrieben. Hierbei wird ein Koordinaten-Tripel, bestehend aus Zylindernummer, Kopfnummer und Sektornummer, benutzt um einzelne Sektoren auf der Platte eindeutig zu lokalisieren. Dieses Verfahren ähnelt im Aufbau Zylinderkoordinaten. Hier beschreiben Radius, Höhe und Winkel einen Punkte eindeutig im Raum.

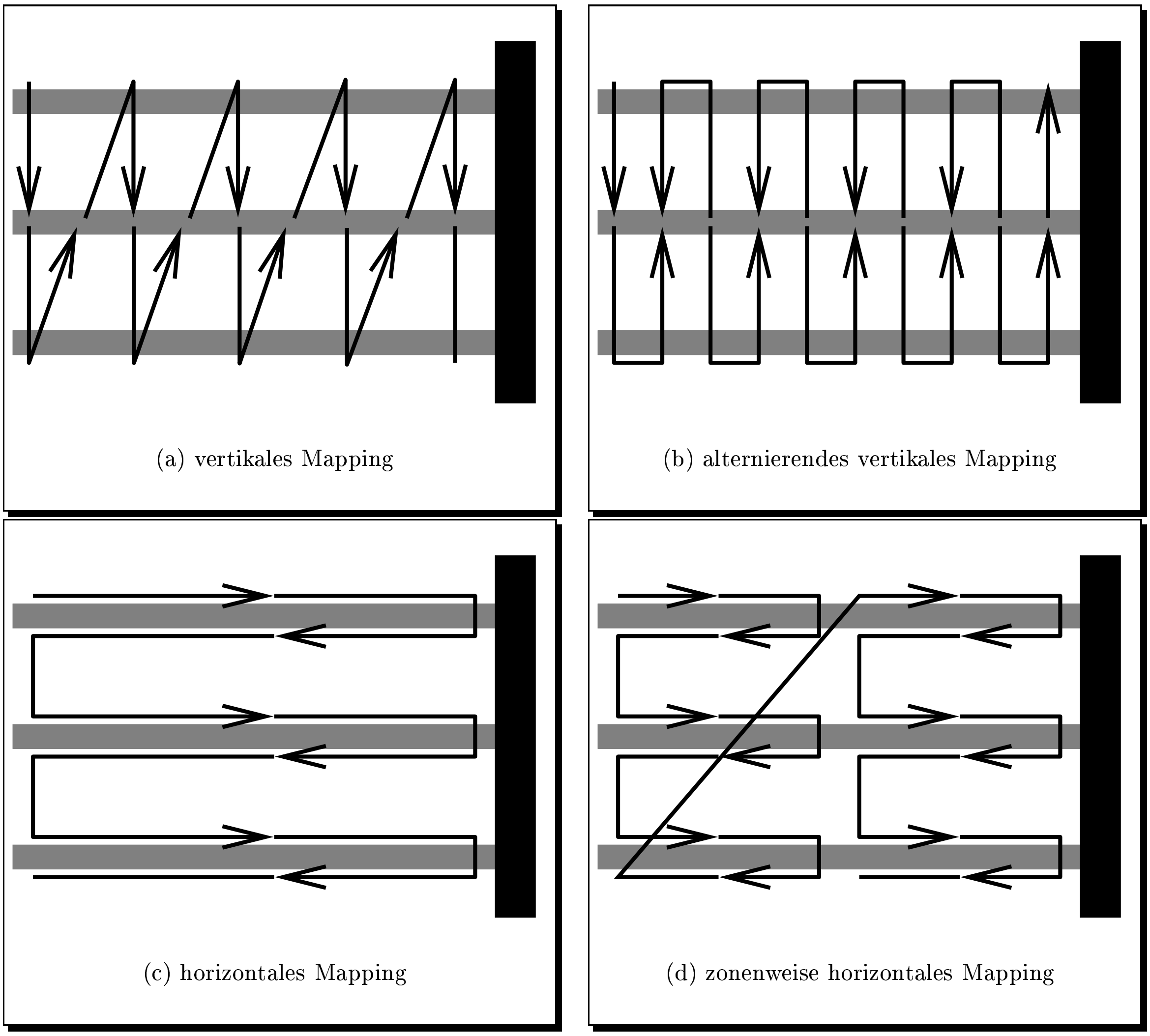
Später wurde das CHS-Verfahren nur noch aus Kompatibilitätsgründen benutzt. Die Platte selbst rechnete die CHS-Koordinaten noch einmal auf die echten physischen Positionen um. [Köh96]

Heutzutage wird fast ausschließlich das LBA-Verfahren (*linear block addressing*) eingesetzt. Die Platte rechnet dabei selber die vom Rechner benutzten logischen Adressen auf physische Positionen um. Auch im Plattentreiber werden nur noch die logischen Adressen benutzt.

Dieses Umrechnen bzw. Abbilden der logischen Adressen auf physische Positionen wird bei Festplatten als Mapping bezeichnet.

2.1.6 Mapping-Arten

Das Mapping findet heutzutage vollständig in den Platten statt. In Abbildung 2.4 sind übliche Mapping-Arten dargestellt.



(a) vertikales Mapping

(b) alternierendes vertikales Mapping

(c) horizontales Mapping

(d) zonenweise horizontales Mapping

Abbildung 2.4: Verschieden Arten des Mappings

Ein heute verbreitetes Verfahren, welches auch auf allen Festplatten, die in dieser Arbeit praktisch vermessen wurden, festgestellt wurde, ist das vertikale Mapping. Logische Adressen steigen dabei von den äußeren Zylindern zum Platteninneren hin an. Wenn das Ende der ersten Spur erreicht ist, wird die nächste Spur im selben Zylinder benutzt, bis alle Spuren im äußersten Zylinder belegt sind. Als nächstes wird der nächste innere Nachbar abgebildet. Dieses Verfahren hat den Vorteil, dass der Kamm bei sequentiellen Zugriffen nur minimal bewegt werden muss. Der Weg, den er zurücklegen muss, verhält sich im wesentlichen[2] proportio-

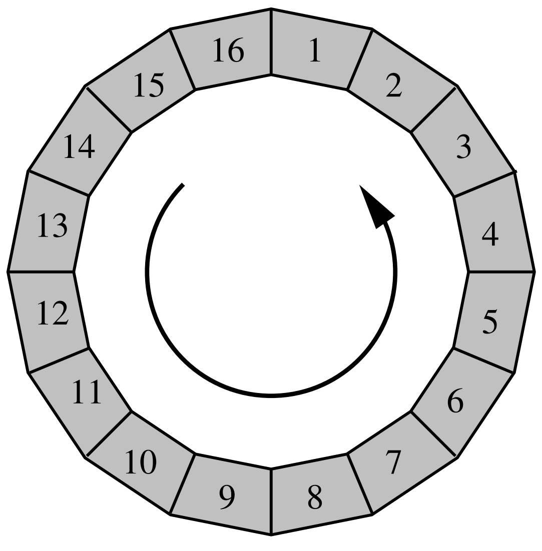
[2] Ausnahmen davon sind sehr kurze Distanzen im Bereich des selben Zylinders und Anomalien, die durch das Remapping von fehlerhaften Sektoren entstehen.

nal zum Abstand der logischen Sektoradressen. Die Platten-Scheduler vieler Betriebssysteme gehen heute beim Umsortieren einzelner Aufträge von einem solchen Mapping aus.

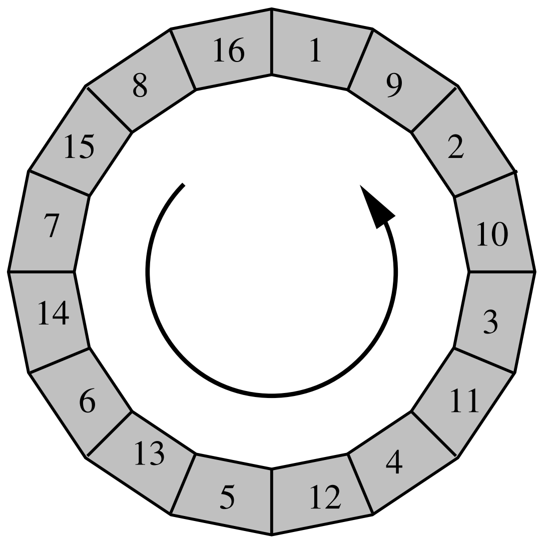
Das alternierende vertikale Mapping, wie in [AAD97] nachgewiesen, unterscheidet sich eigentlich nur in der Reihenfolge in der die einzelnen Spuren eines Zylinders angeordnet werden vom normalen vertikalen Mapping. Wie weiterhin festgestellt wurde, kann die Kopfumschaltzeit durchaus stark vom Quell- und Zielkopf abhängen, muss also insbesondere nicht konstant sein. Insofern kann das alternierende vertikale Mapping den sequentiellen Zugriff auf die Platte optimieren, weil dadurch der Kopfversatz zwischen benachbarten Spuren kleiner eingestellt werden kann.

Beim horizontalen Mapping wird mit ansteigender logischer Sektornummer zuerst eine Oberfläche vollständig belegt bevor zur nächsten gesprungen wird. Diese Anordnung ist bei Festplatten sinnvoll, deren Kopfumschaltzeit größer als die Resettle-Zeit ist. Bei dieser Art von Mapping kann man nicht mehr von klarer Zonentrennung sprechen, da die unterschiedlich schnellen Zonen ja auf jeder Oberfläche auftreten.[Boe96]

Auf sehr alten Platten findet man noch eine andere Art des Mappings (siehe Abbildung 2.5), die als Sector-Interleaving bezeichnet wird. Sie tritt unabhängig zu den bisher beschriebenen auf und betrifft nur die Anordnung der Sektoren innerhalb einer Spur. Grund für diese Anordnung war, dass damalige Kontroller die Daten nicht so schnell verarbeiten konnten, wie die Platte rotierte. Dargestellt ist die Position der Sektoren in der Spur. Für das Auslesen einer ganzen Spur benötigt die Platte bei einfachem Interleaving zwei Umdrehungen des Plattenstapels.



(a) kein Interleaving (heute üblich)



(b) einfaches Interleaving

Abbildung 2.5: Verschiedene Formen des Sector-Interleaving

Für Worst-Case-Untersuchungen ist das verwendete Mapping wichtig. Mit diesem Wissen kann man Worst-Case-Seeks provozieren, die die maximale Laufzeit des Kamms beinhalten. Bei vertikalem Mapping lässt sich das durch abwechselnde Aufträge auf den logischen Plattenanfang und auf das logische Plattenende relativ einfach realisieren (siehe auch Abschnitte 3.2.2 und 4.2.7). Bei horizontalem Mapping sind solche Auftragspaare schwieriger zu erzeugen. Hier müsste man das genaue Mapping und damit die logischen Blocknummern kennen, die in die äußersten und innersten Spuren abgebildet werden.

2.1.7 Datenübertragung

Für die Übertragung der Daten zwischen Platte und Rechner gibt es zwei unterschiedliche Verfahren:

- Beim **PIO-Modus** (*programmed io*) ist der Prozessor dafür verantwortlich jedes einzelne Datum von der Platte abzuholen und in den Speicher zu schreiben (bei lesenden Zugriffen), bzw. andersherum bei Schreibzugriffen. Der Prozessor ist in dieser Zeit vollständig damit ausgelastet, die Daten zu kopieren. Dieses Verfahren wird heute nur noch selten verwendet.

- Für den **DMA-Modus** (*direct memory access*) muss der Plattenkontroller selbst Speicherzugriffe auf den Hauptspeicher beherrschen. Um den Datentransfer einzuleiten, programmiert der Prozessor den Plattenkontroller mit den ihm zugewiesenen Speicherbereich und startet den Datentransfer. Der Kontroller transportiert die Daten selbstständig zwischen Platte und Speicher. Besonders bei Multitasking-Betriebssystemen zeigt sich der Vorteil des DMA-Verfahrens, da die Ressource Prozessor in der Zwischenzeit für andere Aufgaben eingesetzt werden kann.
  Beim Ende eines Transfers löst der Kontroller einen Interrupt aus, den der Prozessor behandelt. In der Interrupt-Behandlungsroutine wird der aktuelle Plattenauftrag abgeschlossen und der nächste vorbereitet und gestartet.

Aktuelle IDE-Platten beherrschen den UDMA-100 bzw. sogar den UDMA-133 Modus, wobei maximal 100 bzw. 133 MByte/s übertragen werden können. Im schnellsten PIO-Modus sind maximal 16,6 MByte/s möglich. Die Auslastung moderner IDE-Platten ist nur im DMA-Betrieb möglich. Die Dauertransferrate z. B. der in dieser Arbeit untersuchten Seagate Barracuda ATA IV (siehe auch Anhang A) liegt in der schnellsten Zone bei knapp 42 MByte/s (siehe Abbildung 2.3) und damit deutlich über dem Maximum der PIO-Modi. Bei Plattenaufträgen, die aus dem Plattencache bedient werden können, ist sogar der derzeit schnellste Übertragungsmodus der limitierende Faktor.

In Abbildung 2.6 ist der prinzipielle Zeitablauf eines lesenden Plattenauftrages im DMA-Modus dargestellt. Die Zeitachse ist nicht maßstabsgerecht. Vorgänge die auf Rechnerseite dargestellt sind brauchen gegenüber den Vorgängen auf Plattenseite bei heutigen Platten nur einen Bruchteil der Zeit. Der Zeitanteil des Gesamtzugriffes, der auf die Positionierung des Kopfes und die Rotationsverzögerung entfällt ist je nach Auftrag unterschiedlich und hängt u. a. von der Position des Kammes, dem Zielsektor und dem Rotationswinkel der Platte zum Startzeitpunkt des Auftrages ab.

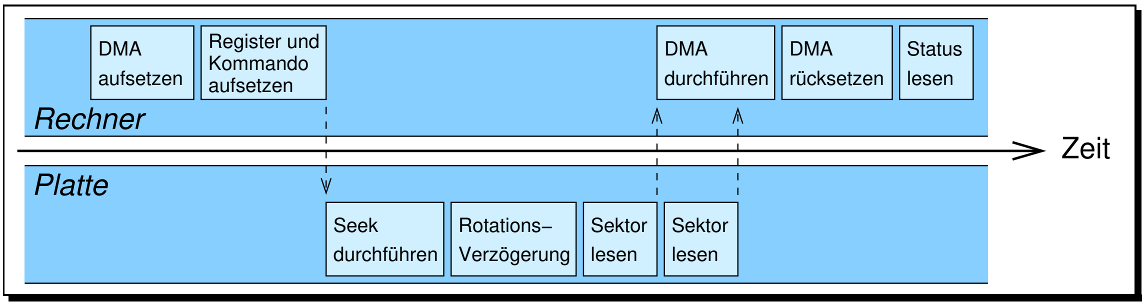


Abbildung 2.6: Lesender Plattenzugriff (DMA) nach [Sch94, Seite 66], (verändert)

2.2 Verwandte Arbeiten

Zu den Themen der Festplattenmodellierung und der Ermittlung von Laufzeitparametern existiert eine Reihe von Arbeiten von denen einige im Folgenden kurz vorgestellt werden.

2.2.1 DIXtrac

`DIXtrac` ist ein Programm, das möglichst automatisch Festplattenbeschreibungen erstellen soll. Es kann laut Entwickler über 100 Parameter automatisch ermitteln und benutzt dazu eine Algorithmensammlung, die teilweise mit Experimenten und teilweise mit dem Auslesen von Parametern aus den *mode-pages* arbeitet. Voraussetzung sind deshalb Festplatten mit SCSI-2 Schnittstelle. Das Ausgabeformat der Ergebnisse ist für den Festplattensimulator DiskSim ([GWP98]) geeignet. Damit können laut [SchGa99] genaue Festplattenmodelle erstellt werden. Während Ganger und Schindler in [GaSch00] zwölf Plattenparametersätze zur Verfügung stellen, wovon sieben mit `DIXtrac` erstellt wurden, ist `DIXtrac` selber nicht verfügbar.

Schindler et al. beschreiben in [SGL02] die Implementierung eines Dateisystems für Free-BSD, welches auf von `DIXtrac` gewonnenen Parametern aufbaut. Speziell wird hier versucht Plattenaufträge an Spurgrenzen und -größen anzupassen um dadurch den Durchsatz zu erhöhen. Außerdem wird auf harte und weiche Echtzeit eingegangen. Allerdings werden keine genauen Zeiten für einzelne Aufträge genannt, sondern die Performance von Videoservern wird verglichen. Es wird angeführt, dass durch die direkte Kontrolle von Seeks und Spurumschaltzeiten auch die Worst-Case-Zeiten reduziert werden können. Nicht gezeigt wird aber, wie konkrete Worst-Case-Zeiten für einzelne Platten vorherzusagen sind, bzw. wie Zugriffsstatistiken für typische Zugriffsmuster auf konkreten Platten zu erzeugen sind.

2.2.2 zcav

`zcav` ([Coker]) Ist ein kleines Programm aus der Festplattenbenchmarksuite `bonnie++`. Das Programm ermittelt die Dauertransferrate in verschiedenen Bereichen von Datenträgern wie z. B. Festplatten, CD-ROMs und Disketten. Bei Festplatten kann man so die Zonenaufteilung nachweisen. Dieses Programm wurde als Ausgangspunkt für erste Experimente zur Bestimmung von Festplattenparametern benutzt. Ein Beispielresultat findet sich in Abbildung 2.3. Für die sektorgenaue Bestimmung von Zonengrenzen ist `zcav` aber ungeeignet.

Meter benutzt `zcav` in [Met97] für Performance-Untersuchungen eines BSD-Dateisystem.

2.2.3 Microbenchmark based Extraction

In [TAP00] stellen Talagala et al. verschiedene Ansätze vor um einzelne Festplattenparameter experimentell zu bestimmen. Dazu werden drei Microbenchmarks benutzt. Das sind `skippy`, `zoned` und `seeker`. Sie erfüllen folgende Aufgaben:

- `seeker` erstellt ein einfaches Seek-Profil, d. h. es extrahiert den Zusammenhang von Seek-Weite und Zeitdauer,
- `zoned` ermittelt die ungefähren Grenzen der Plattenzonen, ähnlich wie das in Abschnitt 2.2.2 vorgestellte `zcav`,
- `skippy` erzeugt durch linear wachsende Abstände zwischen Zugriffspaaren Graphen für einzelne Zonen der Platte, aus denen sich erstaunlich viele Parameter extrahieren lassen.

Bisher ist kein einfacher Weg bekannt die Parameter aus dem Graph automatisch auszulesen. Graphen, die an realen Platten erstellt wurden, können außerdem ein nicht zu vernachlässigendes Rauschen enthalten, welches die automatisierte Weiterverarbeitung zusätzlich erschwert.

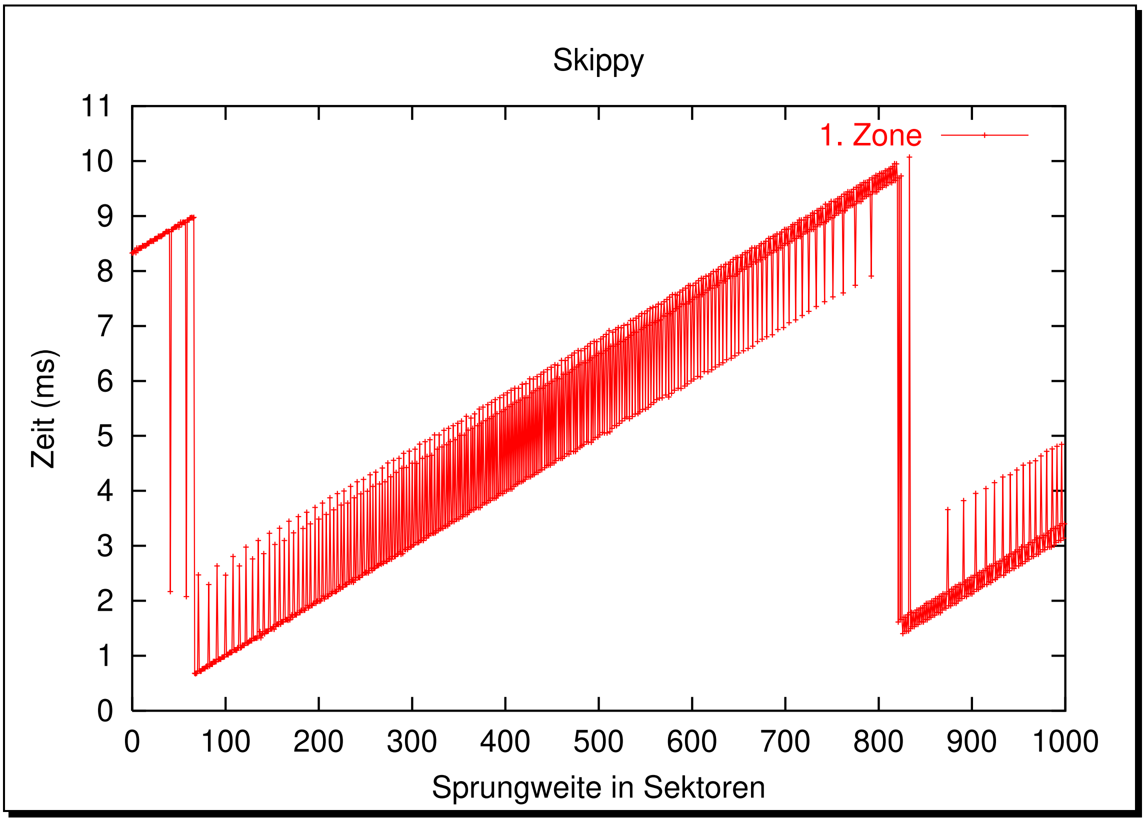


Abbildung 2.7: Ergebnis eines `skippy`-Laufes

Abbildung 2.7 wurde mit einer Implementierung im Rahmen der vorliegenden Arbeit, basierend auf dem Pseudocode aus [TAP00] erzeugt.

2.2.4 Temporally Determinate Disk Access: An Experimental Approach

Ziel der in [AAD97] beschriebenen Arbeit ist es einen Festplattentreiber zu entwickeln, der sich möglichst deterministisch bzgl. der Bearbeitungszeiten von Plattenaufträgen verhält. Die Arbeit stellt eine Sammlung von praktischen Experimenten vor, um einzelne Plattenparameter zu bestimmen. Diese Parameter sollen als Basis für ein genaues Festplattenmodell benutzt werden. Auf harte Echtzeitanforderungen wird in der Arbeit aber nicht näher eingegangen.

2.2.5 Dateisystem-Traces

Ousterhout et al. beschreiben in [OCH85] die Untersuchung des Dateisystems von Unix 4.2 BSD. Es werden grundlegende Erkenntnisse über den Bandbreitenbedarf pro Nutzer, die Verteilung von Dateigrößen und von ihren Lebensdauern sowie typische Arten des Dateizugriffs präsentiert. Die Arbeit zielt auf allgemeine Erkenntnisse ab, um globale Parameter von Dateisystemen, wie etwa die Blockgröße, die Cache-Größe u. ä. sinnvoll beeinflussen zu können. Die Untersuchungen werden dabei nicht auf der Ebene von einzelnen Plattenzugriffen gemacht, sondern es werden bestimmte Operationen auf der Dateisystemebene protokolliert.

Vogels zeigt in [Vog99] ähnliche Ergebnisse für NTFS, das Dateisystem von Windows NT 4.0. Auch hier wird die gleiche Granularität benutzt.

Während das prinzipielle Vorgehen, das Verhalten von Anwendungen bzw. ihren Benutzern auf Dateisystemebene zu überwachen, für die Ziele der genannten Arbeiten sinnvoll erscheint, ist diese Granularität für das Ziel der vorliegenden Arbeit zu grob. Eine Protokollmöglichkeit in einer Schicht unterhalb des Dateisystems, z. B. im Plattentreiber, erscheint hierfür eher geeignet. Damit würden die gemessenen Daten nicht vom Betriebs- und Dateisystem abhängen und auch auf andere Systeme übertragbar sein.

3 Entwurf

3.1 Entwurfsziele

Ziel dieser Arbeit war es, eine Reihe von Werkzeugen zu entwickeln, die das Zeitverhalten von Festplatten möglichst automatisch ermitteln können. Zu lösende Teilprobleme waren dabei:

- die Ermittlung der Worst-Case-Zeiten für Plattenzugriffe,
- die Aufzeichnung von typischen Zugriffsmustern,
- das Abspielen der gewonnenen Zugriffsmuster auf andere Platten,
- die Protokollierung wichtiger Daten — insbesondere der Einzelauftragsbearbeitungszeiten — beim Abspielvorgang und
- die Zeitmessung und Protokollierung der Plattenzugriffen mit einer zeitlichen Mindestauflösung von 0,1 ms.

Im weiteren Verlauf dieses Kapitels wird besonders auf die Bestimmung der Worst-Case-Zeiten und Eingrenzung der Situationen bei denen sie auftreten eingegangen. Am Ende dieses Kapitels wird eine Lösung für das Aufnehmen, das Abspielen und die Protokollierung von Mustern entworfen. Für die Zeitmessung und Protokollierung der Einzelaufträge soll möglichst eine fertige Lösung benutzt werden, ebenso wie für die Plattentreiber und weitere notwendige Infrastruktur. Dieses Vorgehen dient nicht nur der Reduzierung des Implementierungsaufwandes in der Arbeit selbst, als auch vor allem der Minderung des dauerhaften Pflegeaufwandes, der durch den stetigen Fortschritt bei der Festplattenentwicklung notwendig wäre.

3.2 Modell

Um Worst-Case-Zeiten zu ermitteln gibt es zwei prinzipielle Herangehensweisen:

- den Modellansatz und
- die direkte Messung.

Beim **Modellansatz** wird ein Verhaltensmodell mit einzelnen Parametern aufgestellt. Daraus können Situationen, in denen der Worst-Case auftritt, ermittelt werden. Außerdem kann das Zeitverhalten mithilfe des Modells für diese Situationen berechnet werden.

Der Nachteil dieses Ansatzes ist, dass realistische Modelle, die das Zeitverhalten von Platten genau voraussagen sehr komplex sind. Viele Parameter darin lassen sich darüber hinaus nicht ohne Wissen über den inneren Aufbau und das Verhalten der Firmware der Platte bestimmen. Diese Informationen sind jedoch nur dem Hersteller bekannt.

Als Alternative dazu bietet sich die **direkte Messung** von Bearbeitungszeiten an. Diese Methode lässt sich relativ leicht umsetzten. Der Nachteil ist, dass man, um den Worst-Case zu messen, wissen muss, wann er auftritt bzw. wie man ihn provozieren kann. Ansätze, die eine sehr große Menge von zufälligen Aufträgen an die Platte schicken, können zwar zu brauchbaren Resultaten führen, aber eine Garantie, dass ausgerechnet der Worst-Case mit abgedeckt wurde, hat man nicht. Solche Versuche sind extrem zeitaufwendig.

Um die Worst-Case-Zeiten für Festplattenaufträge zu ermitteln wird deshalb ein vereinfachter Modellansatz benutzt. Die einzelnen Parameter müssen durch verschiedene Messungen an der Platte bestimmt werden.

3.2.1 Voraussetzungen

Damit das Modell überschaubar bleibt, und damit auch in der Praxis akzeptable Worst-Case-Zeiten Zustande kommen, müssen einige Randbedingungen definiert werden:

- Das Verhalten des Plattencaches soll nicht mit modelliert werden. Für die Betrachtung von Worst-Case-Zeiten ist nur der Schreibcache der Platte interessant. Er beschleunigt die Ausführung vieler Aufträge bzw. viele Aufträge werden mit aktiviertem Schreibcache vor dem echten Schreiben bestätigt. In dem Moment, in dem der Schreibcache allerdings voll ist und weitere Aufträge ankommen, müssen die zwischengespeicherten erst einmal abgearbeitet werden. Genau in dieser Situation werden Aufträge dann deutlich stärker verzögert, als es ohne Schreibcache der Fall wäre.
  Der Lesecache hat auf die harten Echtzeiteigenschaften im Betrieb keine negativen Auswirkungen, da Informationen daraus einfach verworfen werden können.

- Der Bus über den die Platte an den Rechner angebunden ist, muss für sie im Bedarfsfall immer sofort verfügbar sein. Er darf für die Datenübertragung beim Worst-Case keinen Flaschenhals darstellen. Das bedeutet, dass die Platte entweder als einziges Gerät am Bus betrieben werden muss, oder dass ein zusagenfähiger Bus-Scheduler eingesetzt wird. Beispiele für letzteres Vorgehen finden sich in [BSG98, Meh98].
  Für IDE-Platten, wie sie in der vorliegenden Arbeit untersucht wurden, erscheint der Einsatz einer Platte pro IDE-Kanal als sinnvolle Lösung.

- Es wird weiterhin vorausgesetzt, dass Worst-Case-Zeiten nicht durch Startvorgänge der Platte, physische Fehler, Verzögerung aufgrund thermischer Kalibrierung und das Remapping von fehlerhaften Sektoren entstehen.

3.2.2 Einfaches Modell

Aufbauend auf [BSG98, RueWi94, Scha99, Shr97, WGPW96] und dem zeitlichen Ablauf, wie er in Abbildung 2.6 dargestellt ist, wurde zuerst folgendes einfaches Modell für Plattenaufträge mit einem Sektor entwickelt:

$$time_{wc} = time_{seek} + time_{rotation} + time_{sector-rotation} + time_{overhead} \tag{3.1}$$

Aus der Tatsache, dass nur die Worst-Case-Zeiten zu ermitteln sind, ergeben sich Vereinfachungen gegenüber den Modellen aus der Literatur, welche meist entwickelt wurden um die Bearbeitungszeiten von Aufträgen generell vorherzusagen.

Die Bearbeitungszeit für einen Auftrag setzt sich aus einzelnen Teilzeiten zusammen. Das sind in diesem Modell folgende:

- $time_{seek}$ ist die Zeit, die der Schreib-Lesekopf der Platte braucht um die Zielspur anzufahren, inklusive einer eventuellen Kopf-Umschalt-Zeit. Dieser Parameter wird vom Akustikmanagement beeinflusst, das in modernen IDE-Platten zu finden ist. Hier wird z. B. die Beschleunigung des Kammes reduziert um die Platte leiser zu machen.

- $time_{rotation}$ steht für die Zeit, die die Platte mit Warten auf den Zielsektor verbringt, gemessen vom Abschluss des Seeks bis zum Zeitpunkt an dem der Zielsektor unter dem Kopf angekommen ist.

- $time_{sector-rotation}$ ist die Zeit, die ein Sektor braucht, um vollständig unter dem Kopf entlang zu rotieren. Diese Zeit ist nicht für jeden Sektor gleich groß, da die Sektoren in den äußeren Spuren kleinere Winkel einnehmen (-> Zoned-Bit-Recording).

- $time_{overhead}$ enthält die Verzögerung, die durch den Kontroller, Datenübertragungen über den Bus und die Zugriffszeiten auf Speicher etc. entsteht. Diese Zeit wird im Modell als konstant angenommen ([RueWi94]). Dieser Parameter enthält außerdem die Zeit, die zur Datenübertragung zwischen Platte und Hauptspeicher benötigt wird. Wenn der Übertragungsmodus zwischen Plattencache und Rechner jedoch höhere Datenraten erlaubt, als der zwischen Plattencache und Speichermedium, so spielt diese Zeit in der Praxis fast keine Rolle.

Für jeden dieser Parameter muss nun der Worst-Case bestimmt und die entsprechende Zeit berechnet oder gemessen werden. Dazu wurden eine Reihe von Algorithmen entwickelt bzw. aus der Literatur übernommen.

Ein zentraler Begriff ist hier die sogenannte *MTBRC* (*minimum time between request completion*) [WGPW96]. Hier wird z. B. die Zeit zwischen zwei Festplattenaufträgen gemessen. Bei der vorliegenden Arbeit wurde fast immer die Zeit zwischen zwei Schreibzugriffen benutzt und dabei meist vom Ende des ersten bis zum Ende des zweiten Auftrages gemessen. Der erste Teilauftrag wird oft benutzt um einen definierten Startzustand (Zylinder, Winkel, aktiver Kopf) für den zweiten Teilauftrag zu schaffen.

Bestimmung der Worst-Case-Zeiten

Wie schon weiter oben angedeutet, muss für jeden Parameter des Modells die Worst-Case-Situation festgestellt und der entsprechende Wert ermittelt werden:

- Der Worst-Case für $time_{seek}$ tritt typischerweise dann auf, wenn der Kamm den längsten Weg zurücklegen muss, d. h. von ganz innen nach ganz außen bzw. andersherum (Full-Stroke-Seek) bewegt wird. Dieser Fall tritt bei Zugriffspaaren auf, die Sektoren aus den ersten und letzten Zylindern enthalten. Bei linearem Mapping entsprechen diese beiden Positionen den ersten und den letzten logischen Blöcken der Platte.

- Für den Parameter $time_{rotation}$ tritt der Worst-Case auf wenn der Kopf den Beginn des Zielsektor gerade verpasst hat, und somit eine volle Rotation warten muss bis der Zielsektor wieder vorbei kommt. Dieser Parameter entspricht damit der normalen Umdrehungszeit der Platte. Bei einer Platte mit 7.200 u/min wären das beispielsweise 8,3 ms.
- Der Parameter $time_{sector-rotation}$ ist an sich konstant, aber je nach Zone unterschiedlich. Er ist in den innersten Zylindern der Platte am größten, da ein Sektor dort den größten Winkel einnimmt und damit am längsten unter dem Kopf hindurch rotiert.
- Da $time_{overhead}$ im Modell als konstant angenommen wird, müssen keine besonderen Worst-Case-Betrachtungen gemacht werden.

3.2.3 Verbessertes Modell

In den praktischen Versuchen hat sich gezeigt, dass das einfache Modell keine vollständige Vorhersage ermöglicht.

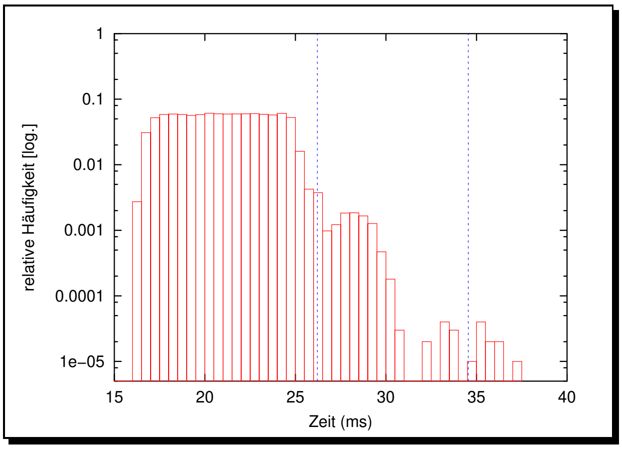


Abbildung 3.1: Histogramm der Abarbeitungszeit von 100.000 Worst-Case-Schreibzugriffen mit einem Sektor

In Abbildung 3.1 ist das Histogramm der Abarbeitungszeiten von Worst-Case-Plattenaufträgen dargestellt. Um diese zu generieren wurden immer abwechselnd im Anfangs- und im Endbereich der Platte ein Sektor geschrieben, sodass jeweils ein maximaler Seek auftritt.

Ein bestimmter Anteil der Aufträge wird nicht innerhalb der mit dem einfachen Modell 3.1 ermittelten Zeitgrenze (hier ca. 26 ms) abgeschlossen. Weiterhin ist zu erkennen, dass der größte Anteil derjenigen Aufträge, die nicht rechtzeitig fertig werden, innerhalb einer weiteren Umdrehung der Platte (hier bei ca. 34 ms) beendet wird.

Es gibt eine Reihe von bekannten Faktoren, die zu längeren Bearbeitungszeiten führen können:

- mechanische Erschütterungen,
- Schreib-/Lesefehler auf dem Medium (fehlererkennende und -korrigierende Codes werden eingesetzt),

- Seek trifft Zielspur nicht genau,
- Settle braucht zu lange,
- Übertragungsfehler auf dem Kabel,
- Schwankung der Versorgungsspannung.

Diese Fehler können meist durch Wiederholungen kompensiert werden, wobei natürlich auch wieder Fehler auftreten können. Wurde nun durch den erhöhten Zeitbedarf der Zielsektor verpasst, muss mindestens eine weitere Rotation gewartet werden, bevor der Medienzugriff erneut durchgeführt werden kann. Das erweiterte Modell muss also mit einer größeren Worst-Case-Zeit durch weitere Rotationen rechnen:

$$time_{wc} = time_{seek} + n * time_{rotation} + time_{sector-rotation} + time_{overhead} \tag{3.2}$$

Um den Parameter n zu bestimmen wurde ein Experiment entwickelt, das dessen Häufigkeitsverteilung ermittelt. Dazu werden, ähnlich wie bei der Gewinnung der Daten für Abbildung 3.1, eine sehr große Anzahl von Worst-Case-Aufträgen an die Platte geschickt. Die gemessenen Zeiten werden in Bereiche unterteilt, deren Grenzen sich aus der Worst-Case-Zeit aus Modell 3.1 und der Umdrehungszeit der Platte ergeben. Die Implementierung wird in Abschnitt 4.2.9 beschrieben.

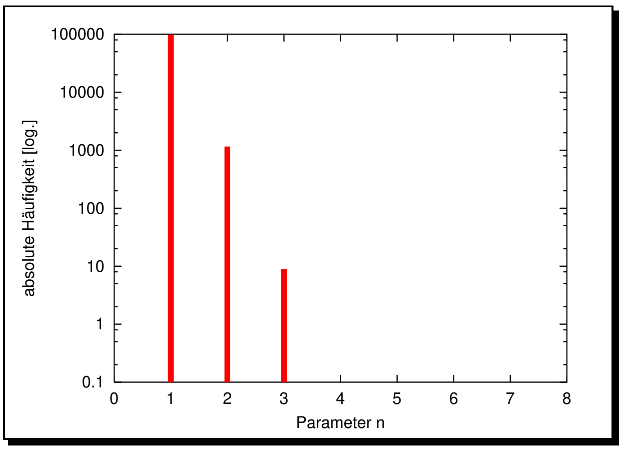


Abbildung 3.2: Verteilung des Parameters n

Die Verteilung für n bei der Zugriffsfolge aus Abbildung 3.1 ist in Abbildung 3.2 dargestellt. Von den 100.000 Einzelaufträgen wurden 98.741 (98,741 %) mit $n = 1$, d. h. ohne zusätzliche Rotationen, ausgeführt. 1.249 (1,249%) Zugriffe benötigten eine Rotation mehr ($n = 2$) und 10 (0,01%) Zugriffe benötigten zwei zusätzliche Rotationen.

Für den Worst-Case muss in dem dargestellten Beispiel also $n = 3$ angenommen werden.

3.2.4 Erweiterung für größere Aufträge

Im normalen Betrieb sind Festplattenaufträge mit einem Sektor eher selten. Typische Größen sind 4 KByte, 8 KByte oder noch größere Vielfache der Seitengröße des Hauptspeicher[1].

[1] meist 4 kByte bei aktuellen i86-Architekturen

Auch die Read-Ahead-Strategie des Betriebssystems selbst kann die Auftragsgröße beeinflussen. Um das Modell für solche Aufträge anzupassen muss man im allereinfachsten Fall einen Faktor für $time_{sektor-rotation}$ vorsehen, da auf mehrere hintereinanderliegende Sektoren zugegriffen wird:

$$\begin{aligned} time_{wc} &= time_{seek} + n * time_{rotation} \\ &\quad + m * time_{sector-rotation} + time_{overhead} \end{aligned} \tag{3.3}$$

m wäre für einen 4 KByte Auftrag bei einer typischen Sektorgröße von 512 Byte also 8. Diese einfache Erweiterung ist aber nicht ausreichend, da ein solcher Auftrag auch über eine Spurgrenze hinweg auftreten kann. Hier muss also noch der Spurversatz (siehe auch Abschnitt 2.1.3) mit berücksichtigt werden.

Bei den Versuchen wurden keine Zugriffe mit mehr als 255 Sektoren beobachtet. Größere Aufträge, die manuell an das *raw-device* geschickt wurden, hat der IDE-Treiber von Linux in 255-Sektoren-Teile und einen Rest aufgeteilt. SCSI-Platten und moderne IDE-Platten beherrschen jedoch Auftragsgrößen mit deutlich mehr als 255 Zielsektoren [ATA01]. Es scheint sich hierbei nur um eine Limitierung durch den aktuellen Linux-Treiber zu handeln. Insofern erscheint es sinnvoll auch mehr als einen maximalen Spurversatz pro Auftrag anzunehmen. Das so erweiterte Modell wird also noch um den Summand $v * time_{max-skew}$ ergänzt:

$$\begin{aligned} time_{wc} &= time_{seek} + n * time_{rotation} + m * time_{sector-rotation} \\ &\quad + time_{overhead} + v * time_{max-skew} \end{aligned} \tag{3.4}$$

Der Faktor v muss dabei jeweils nach Zahl der Zielsektoren im Auftrag und Sektoren pro Spur in der innersten Plattenzone bestimmt werden. In Abbildung 3.3 ist das Ergebnis von zwei praktischen Versuchen, in denen immer abwechselnd im Anfangsbereich und im Endbereich der Platte Zugriffe durchgeführt werden, dargestellt.

Jeder Teilauftrag ist 64 KByte und damit 128 Sektoren groß. Dargestellt ist das Histogramm für die Bearbeitungszeiten sowohl für schreibende, als auch für lesende Zugriffe. Sehr deutlich kann man sehen, dass ein überwiegender Teil der Zugriffe innerhalb der Grenze des einfachen Modells aus Abschnitt 3.2.2 erfolgt (hier bei 31 ms). Innerhalb der Zeit für eine weitere Umdrehung der Platte werden auch fast alle verbleibenden abgeschlossen.

Um die Vermutung zu überprüfen, ob Schreibzugriffe länger dauern, als Lesezugriffe, wie in Abschnitt 2.1.2 vorausgesetzt, wurde der selbe Versuch auch mit lesendem Zugriff ausgeführt. Dabei bleibt jedoch der Lesecache eingeschaltet (er lässt sich bei vielen Platten nicht abschalten). Es ist eine starke Häufung von Zugriffszeiten fast um 0 ms zu erwarten, da ein Teil der Aufträge direkt aus dem Cache der Platte beantwortet werden kann. Weiterhin wird sich das Hauptplateau, welches schon bei dem Versuch mit Schreibzugriffen zu beobachten war, etwas weiter zum Nullpunkt hin verschieben, da bei vielen Aufträgen Teilantworten aus dem Cache erfolgen können, und die entsprechenden Aufträge deswegen etwas schneller ausgeführt werden.

Wenn man die beiden zu erwartenden Veränderungen vernachlässigt, ergibt sich eine ähnliche Situation wie bei schreibendem Zugriff. Der Worst-Case wird von Schreibaufträgen erzeugt.

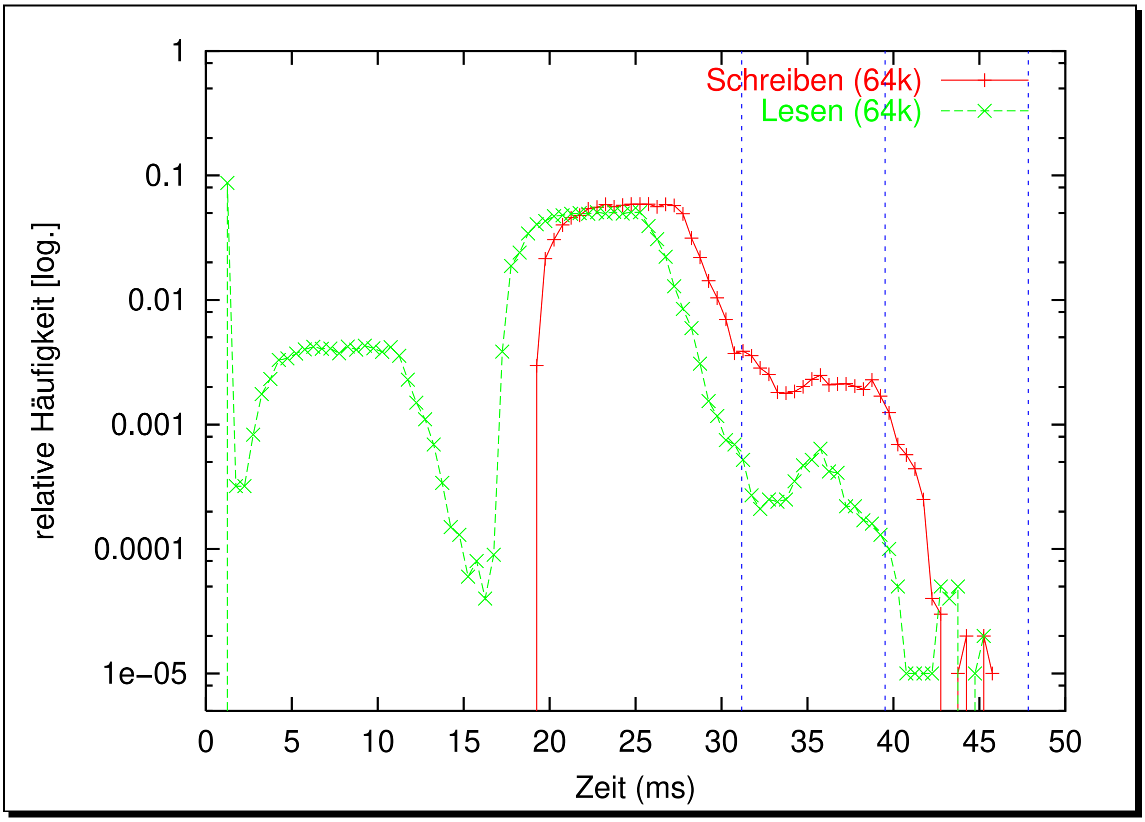


Abbildung 3.3: Bearbeitungszeiten für 100.000 64 KByte große Aufträge

3.2.5 Erweiterung für Fehlermapping

Der Kontroller stellt gegenüber dem Rechner eine kontinuierliche, lückenlose Menge an Blöcken bereit. Fehlerhafte Sektoren auf der Platte werden ausgeblendet. Diese Tatsache hat zur Folge, dass zwei logisch aufeinander folgende Blöcke nicht physisch aufeinander folgenden Sektoren zugeordnet sein müssen. Infrage kommen folgende Verfahren [SGL02]:

- Fehlerhafte Sektoren werden ausgeblendet (*slipping*) und die logischen Blöcke $(n, n+1)$ werden auf die physischen Sektoren $(m, m+1+f)$ abgebildet. Dabei ist f die Anzahl der ausgeblendeten fehlerhaften Sektoren.
  Um die Ausbeute bei der Festplattenherstellung zu erhöhen werden nicht nur vollkommen fehlerfreie Platten in den Geräten verbaut. Bei dem nach der Herstellung folgenden Low-Level-Formatieren werden die fehlerhaften Sektoren ausgeblendet. Die Low-Level-Formatierung kann im Prinzip jederzeit wiederholt werden, wobei aber alle Daten auf der Platte verloren gehen.

- Den fehlerhaften Sektoren werden Ersatzsektoren zugewiesen (*remapping*), die in einem Reservebereich liegen. Das kann das Spurende sein, an dem bei einigen Platten Ersatzsektoren reserviert werden. Das könnte die nächstliegende Ersatzspur sein oder aber irgendein beliebig weit entfernter Reservesektor auf der Platte. Das genaue Vorgehen liegt in der Hand des Herstellers.

> Dieses Verfahren wird typischerweise im laufenden Betrieb eingesetzt, da nicht alle Sektoren verschoben werden müssen und für fast alle Sektoren die Zuordnung von logischer und physischer Adresse bestehen bleibt.

Auf Grundlage des eben dargelegten Sachverhaltes und unter der ungünstigsten Annahme, dass jeder einzelne Sektor eines Plattenauftrages mit einem maximalen Seek erreicht werden muss, stellt sich das Modell nun so dar:

$$\begin{aligned} time_{wc} &= m * (time_{seek} + n * time_{rotation} + time_{sector-rotation}) \\ &\quad + time_{overhead} \end{aligned} \tag{3.5}$$

Das bedeutet, dass ein Zugriff auf m Sektoren fast m-mal so lange dauern kann wie ein Zugriff auf einen Sektor. Dieses ungünstige Verhalten konnte aber in den praktischen Versuchen nie beobachtet werden.

Die Aussagen dieses Modells wären wahrscheinlich für den praktischen Gebrauch deutlich zu pessimistisch und unbrauchbar. Um das Modell hier sinnvoll anzupassen ist mehr Wissen über die *Remapping*-Strategien von Platten notwendig. Außerdem müsste man Platten kontrolliert leicht beschädigen, um mit diesen Fehlern dann das zeitliche Verhalten beobachten zu können. Schindler et al. erwähnen in [SGL02] eine Möglichkeit das Remapping zu erkennen, die allerdings SCSI-spezifisch ist.

Aus diesen Gründen wird im folgende verstärkt das Modell aus Abschnitt 3.2.4 weiter betrachtet.

3.3 Verteilung und Zugriffsmuster

Zur Erzeugung von typischen Zugriffsmustern bietet sich die Beobachtung realer Anwendungen an. Die Zugriffsmuster der Anwendungen werden im laufenden Betrieb mitprotokolliert. Dabei müssen alle relevanten Informationen, die zur Wiederholung der Muster dienen mit gesichert werden. Die aufgezeichneten Muster werden später auf verschiedenen Platten wieder abgespielt. Dabei wird dann das Zeitverhalten der Platte beobachtet. Die Zugriffe beim Abspielen sollten direkt auf die Platte erfolgen, also insbesondere ohne zusätzliche Caches außerhalb der Platte.

Die Zeiträume zwischen zwei Plattenaufträgen sollen bei der Aufzeichnung und beim Abspielen der Muster vernachlässigt werden. Das hat folgende Gründe:

- Die Zeiträume zwischen zwei Plattenaufträgen hängen hochgradig vom benutzten Rechner[2]ab. Das Ergebnis dieser Messung soll jedoch möglichst nur von der Platte abhängen.
- Es ist nicht sofort klar, ob sich das spätere Abschicken eines Auftrages auf dessen Bearbeitungszeit bei eine konkreten Platte positiv oder negativ auswirkt. Späteres Abschicken kann die Bearbeitung des Auftrages verkürzen, da die Platte z. B. nicht so lange bis zum Zielsektor drehen muss, aber sie auch verlängern, wenn der Zielsektor

[2] Ein schnellerer Prozessor kann z. B. dazu führen, dass in einer Anwendung eine Berechnung zwischen zwei Plattenzugriffen wesentlich schneller erfolgt und die gesamte Bearbeitungszeit damit eine Plattenrotation kürzer ist.

durch das spätere Abschicken gerade verpasst wurde. Iyer und Druschel schlagen in [IyDr01] vor im Platten-Scheduler künstliche Verzögerungen einzufügen, um dadurch die Gesamtperformance des Systems zu steigern. Dabei wird ausgenutzt, dass Programme ihre sequentiellen Zugriffe oft durch kurze Rechenpausen unterbrechen. Wird kurz auf den nächsten Auftrag gewartet, kann die Gesamtzeit, die die Platte mit Seeks verbringt, reduziert und damit ihre Auslastung erhöht werden.

- Es sollen vor allem Situationen mit gefüllter Plattenauftragswarteschlange im Vordergrund der Betrachtung stehen, denn diese Situation ist typisch für hohe Last. Dabei treten keine Wartezeiten zwischen Aufträgen auf.

Der Vorteil gegenüber klassischen Anwendungsbenchmarks ist die schnelle und automatische Durchführbarkeit dieser Methode auf ganz unterschiedlichen Rechnern. Die Muster müssen nur einmal aufgezeichnet werden und sind dann mit vielen Rechnerkonfigurationen mit ähnlichem Ergebnis abspielbar. Sie spiegeln dabei die Auftragssituation auf dem aufzeichnenden Rechner wieder.

Ein ganz ähnlicher Ansatz wird auch von Bremer und Schuster in [BrSch02] beschrieben. Bei der Umstellung des schon lange benutzten `hdbench` wurde ein Anwendungsindex eingeführt, bei dem die Plattenzugriffsmuster bestimmter Anwendungen aufgezeichnet werden. Hierzu wird ein Treiber für Windows XP eingesetzt. Die gewonnenen Muster werden später auf anderen Platten abgespielt. Dabei wird das Zeitverhalten gemessen und ein Leistungsindex berechnet.

4 Implementierung

4.1 Messumgebung

4.1.1 Voraussetzungen

Für die in Abschnitt 3.2 vorgestellten Modelle werden einige Randbedingungen definiert. Diese sollen hier genauer beschrieben und ihre Durchsetzung erläutert werden.

- Der Plattencache hat zwei Auswirkungen, die bei der Implementierung berücksichtigt werden müssen:
  - Er verhindert genaue Messungen einzelner Parameter durch das Verstecken der dahinterliegenden Hardware. Bei den Messungen muss der Cache ausgeschaltet sein. Da viele Platten das Kommando zum Ausschalten des Lesecaches nicht implementieren und da Schreibzugriffe im allgemeinen etwas länger dauern (siehe auch Abschnitt 2.1.2), werden bei der Bestimmung der Worst-Case-Zeiten fast nur Schreibzugriffe benutzt. Der Schreibcache kann bei praktisch allen Festplatten ausgeschaltet werden.
  - Der Schreibcache kann die Worst-Case-Zeiten dramatisch verschlechtern, was sein Ausschalten nicht nur bei den Messungen, sondern auch im Echtzeitbetrieb rechtfertigt.
- Bei den Messungen der einzelnen Parameter für die Modelle sollte die Platte allein an einem Bus betrieben werden. Außerdem sollte eine starke Belastung des Rechners durch andere Programme vermieden werden, welche die Pfade zwischen Hauptspeicher und Platten blockieren könnten.
- Hardwarefehler und sonstiges
  - Die Platte muss erschütterungsfrei, schwingungsfrei[1] und genügend gekühlt betrieben werden.
  - Das Anschlusskabel zwischen Platte und Rechner sollte für die Betriebsart geeignet, nicht zu lang und entsprechend abgeschirmt sein (z. B. benötigt der UDMA-100 Modus ein 80-adriges Kabel mit getrennten Masseleitungen für jede Signalleitung).

[1] Gummieinbaurahmen werden oft benutzt, um die Lautstärke der Platte zu dämpfen. Die akustische Leitung der Seek-Geräusche zum Rechnergehäuse soll damit unterbrochen werden. Diese Konstruktion verhindert aber auch die Wärmeableitung über das Rechnergehäuse. Außerdem können die Seek-Zeiten durch auftretende Schwingungen beeinträchtigt werden.

- Das zusätzliche Belasten des Rechners mit anderen Programmen während der Messung blockiert nicht nur die Speicherpfade, sondern kann auch längere Interruptsperren im Kern des Betriebssystems auslösen. Das sollte vermieden werden, da dadurch die an Interrupts gekoppelte Zeitmessung beeinträchtigt wird.
- Das Akustikmanagement, welches auf modernen IDE-Platten zu finden ist, sollte ausgeschaltet sein, damit die kürzesten Seek-Zeiten[2] erreicht werden. Viele Hersteller bieten entsprechende Programme an, die meist auch mit anderen Platten funktionieren. Die Einstellung auf der Platte ist dauerhaft, muss also nur einmal vorgenommen werden.
- Einige moderne Platten (z. B. das Modell DiamondMax 60 GB vom Hersteller Maxtor) haben bei Auslieferung die Eigenschaft „Write-Verify" aktiviert, wodurch direkt nach jedem Schreibzugriff ein vergleichendes Lesen ausgeführt wird. Um diese Sicherheitskontrolle durchzuführen muss mindestens eine weitere Rotation gewartet werden. Diese Eigenschaft sollte deaktiviert werden, sowohl um die Platte genau vermessen zu können, als auch für den Echtzeitbetrieb. Sie ist nur für Anwendungen mit besonders hohen Anforderungen bzgl. der Datenintegrität gedacht. [Max01]
- Das Powermanagement der zu vermessenden Platten sollte ausgeschaltet sein. Speziell wird davon ausgegangen, dass die Worst-Case-Zeit nicht durch das Hochfahren der Platte entsteht. Für IDE-Platten können entsprechende Einstellungen unter Linux mit `hdparm` beeinflusst werden.
- S.M.A.R.T. und ähnliche Technologien ermöglichen, dass die Platte relativ transparent auf Sektorfehler reagiert und die entsprechenden Blöcke z. B. in Ersatzbereiche verlagert (*Remapping*). Diese Eigenschaft ist für den Echtzeitbetrieb natürlich nicht besonders vorteilhaft und sollte abgeschaltet werden, da solche Verlagerungen sehr lange dauern können.
- Die Verzögerung durch thermische Rekalibrierung sollte bei modernen Platten keine Rolle mehr spielen. Hier wird entweder sofort bei Eingang eines Auftrages die Kalibrierung abgebrochen und der aktuelle Auftrag bearbeitet, oder aber die Platte braucht keine Kalibrierung mehr, da sie die Seek-Tabellen während des Betriebes aktuell hält und auf eingebettete Servo-Informationen auf den Medien zurückgreift. Ursache für die Notwendigkeit der thermischen Kalibrierung ist die Veränderung der Größenverhältnisse in der Platte und Veränderung des Reibungswiderstandes von Kammbewegungen. [Boe97]
- Lineares Mapping (siehe auch Abschnitt 2.1.6) wird für die Messungen vorausgesetzt damit Worst-Case-Zugriffe einfach erzeugt werden können.

4.1.2 Warum Linux?

Die aktuelle Implementierung umfasst eine Reihe von Werkzeugen, die unter Linux lauffähig sind. Obwohl die Wahl dieser Zielplattform den Nachteil hat, dass genaue zeitkritische Messungen in einem Multitaskingbetriebssystem nicht einfach sind, überwiegen die vielen Vorteile gegenüber einer Lösung für ein Nicht-Multitaskingbetriebssystem wie z. B. DOS:

[2] Beim aktiviertem Akustikmanagement werden die Köpfe bei Seeks nicht so stark beschleunigt wie im normalen Betrieb. (vgl. z. B. [BrSch02])

- Linux hat fertige, funktionierende Plattentreiber, die auch aktuelle, schnelle Übertragungsmodi unterstützen.
- Diese Treiber werden weiterentwickelt und an aktuelle Neuerungen angepasst.
- Eine Lösung für sehr genaue Zeitmessung ist vorhanden (siehe auch Abschnitt 4.1.4).
- Der Quellcode für den Linux-Kern liegt offen (Open Source). Damit sind Einblicke in die genaue Funktionsweise und auch Änderungen relativ leicht möglich.
- Die restliche notwendige Infrastruktur — z. B. die Möglichkeit Anwendungen zu starten, die aufzunehmende Zugriffsmuster erzeugen — ist gegeben, da es sich bei Linux um ein vollwertiges, modernes Betriebssystem handelt.

4.1.3 Plattenzugriff

Wenn man versucht auf Blockebene mit Platten zu kommunizieren hat man unter Linux den Vorteil, dass für die Platten Gerätedateien zur Verfügung stehen. Mit Hilfe dieser Dateien kann man aus normalen Anwendungsprogrammen heraus direkt auf die Platte zugreifen, ohne dass ein Dateisystem dazwischen geschaltet ist.

Auf dem Weg vom Anwendungsprogramm zur Platte geht ein normaler Auftrag durch zwei Caches, den von Linux und den von der Platte selber. Sie verzögern bzw. unterbinden Zugriffe unter Umständen, wodurch eine genaue Zeitmessung unmöglich wird. Für jeden Cache muss ein Weg gefunden werden ihn bei den Messungen zu umgehen.

Der IDE-Plattencache kann unter Linux mit Hilfe des Kommandos `hdparm -w 0 /dev/hd?` abgeschaltet werden.

Der Puffercache von Linux lässt sich nicht so einfach abschalten wie der Plattencache. Abhilfe schaffen hier *raw-devices* [Gil02]. Sie ermöglichen ungepufferten Zugriff auf Blockebene. Blockgerätedateien[3] lassen sich in *raw-devices* kapseln. Unter Linux kann man ein solches *raw-device* z. B. durch das Kommando `raw /dev/raw1 /dev/hdd` erzeugen. Zugriffe auf `/dev/raw1` und `/dev/hdd` gehen damit an die Slave-Platte am zweiten IDE-Kanal, direkt bzw. durch den Puffercache.

Man sollte vermeiden auf ein solches Gerät gleichzeitig über das *raw-device* wie auch über den normalen Zugang zuzugreifen, da hier keinerlei Synchronisation erfolgt.

Ein weiterer Vorteil der *raw-devices* ist, dass ein Umkopieren der zu lesenden bzw. zu schreibenden Daten in den Kernspeicher nicht erfolgen muss, wenn die Platte DMA beherrscht. Die Daten werden direkt zwischen Plattenkontroller und Anwenderprozess ausgetauscht. Der Prozess stellt dafür einen Pufferspeicher zur Verfügung, der im Hauptspeicher auf Sektorgröße der Platte ausgerichtet (*aligned*) ist.

Abbildung 4.1 stellt die Wege von Plattenzugriffen ausgehend von Anwenderprozessen, durch den Betriebssystem-Kern bis zur Platte selbst dar.

4.1.4 Zeitmessung

Zur genauen Zeitmessung wird ein Kernpatch[4] benutzt, der eine Schnittstelle für ein ladbares Kernmodul bereit stellt. Dieses Modul stellt über eine Gerätedatei die genauen Zeitpunkte

[3] Dazu gehören auch die von den Festplattentreibern bereitgestellten Gerätedateien.
[4] Dieser Kernpatch wurde als Grundlage für diese Arbeit in einem Komplexpraktikum entwickelt [Poh01].

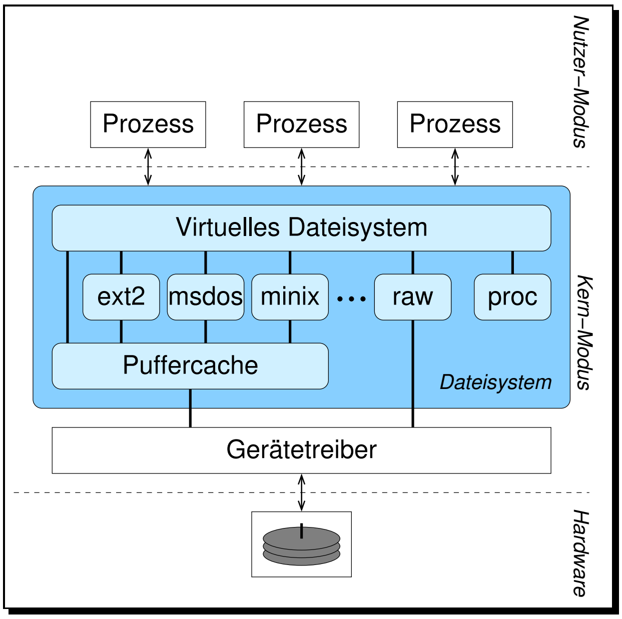


Abbildung 4.1: Dateisystemschichten (nach [BBD95, Seite 158], verändert)

für den Beginn und das Ende von Plattenaufträgen zur Verfügung. Als Zeitbasis wird das TSC-Register[5] (*time stamp counter*) des Prozessors benutzt. Dadurch wird eine für diese Zwecke hinreichende Zeitauflösung im Bereich der Taktfrequenz des verwendeten Prozessors erreicht. Außerdem erfolgt das Auslesen der Zeitwerte mit einem Overhead von nur wenigen Takten und beeinflusst den Rechner so nur minimal.

Über `gettimeofday()` aus `<sys/time.h>` ist diese genaue Zeitmessung aus normalen Prozessen heraus auch möglich, da im Linux-Kern nach Möglichkeit die selbe Zeitquelle genutzt wird [BoCe01, SchGa99]. Bei Aufruf dieser Funktionen sind aber noch zwei Kontextwechsel und die Gefahr der Verdrängung des messenden Prozesses vorhanden. Mit Hilfe des hier genutzten Moduls können auch auf einem relativ stark ausgelasteten Rechner noch genaue Messungen vorgenommen werden, da die Aufzeichnung der Zeitpunkte an Interrupts gekoppelt ist.

Ein Problem könnte hier höchstens die kurzzeitige Sperrung der Interrupts im Kern sein, welche zu kleinen Ungenauigkeiten führen kann.

Das Kernmodul liefert nicht nur die Zeitpunkte der Festplattenaufträge, sondern auch die Sektornummer, die Größe, den genauen Startzeitpunkt, den Auftritt des Abschlussinterrupts und evtl. Zwischeninterrupts, wie sie bei langsameren Übertragungsmodi von IDE auftreten. Mit diesen Informationen kann ein normales Programm Aufträge an die Platte schicken und mit Hilfe des Moduls die genauen Zeitpunkte ermitteln. Das prinzipielle Vorgehen sieht so aus:

[5] Das TSC-Register ist bei Intel-Prozessoren ab dem Pentium, bei AMD-Prozessoren ab dem K5 und bei Cyrix-Prozessoren ab dem 6x86MX implementiert. Es ist damit praktisch auf jeder aktuellen x86-Plattform verfügbar.

```
// Geräte öffnen
f_disk = open(disc_file);
f_time = open(timing_device);

// Plattenauftrag absetzen
lseek(f_disk, sector))
read/write(f_disk, buffer, size * SECTOR_SIZE);

// Zeitinformationen dazu auslesen
read(f_time, &t_start, sizeof(t_start));
read(f_time, &t_end, sizeof(t_end));

// Informationen benutzen
t_start.sector == sector;
t_end.sector   == sector;
t_start.size   == size;

// Zeiten umrechnen und ausgeben
double time = convert_tick_to_time(t_start.time, t_end.time);
cout << time;
```

Der genaue Startzeitpunkt eines Plattenauftrages ist dabei nicht der Moment, in dem ein Prozess mit dem read() bzw. write() Systemruf in den Betriebssystemkern übertritt, sondern der Zeitpunkt, in dem der Auftrag direkt vom Plattentreiber an die Platte übermittelt wird. Die vom Modul bereitgestellten Zeiten liegen mit 64-Bit Genauigkeit[6] in der Einheit „Prozessortakte seit dem Starten des Rechners" vor. Die Umrechnung in normale Zeiteinheiten muss im messenden Programm erfolgen. Grund dafür ist die Entwurfsentscheidung für das Zeitmessungsmodul, den zu messenden Vorgang durch die Messung möglichst wenig zu beeinflussen. So muss im Kern nur das Register ausgelesen werden. Das Modul verwaltet einen Ringpuffer, dessen Inhalt über eine Gerätedatei für den messenden Prozess zugänglich ist.

4.2 Einzelne Algorithmen zur Bestimmung des Worst-Case

In Abschnitt 3.2 wurde ein Modell für die Bestimmung der Worst-Case-Bearbeitungszeiten von Plattenaufträgen entwickelt. Die Vorgehensweise zur Bestimmung der einzelnen Parameter für das Modell sollen in diesem Abschnitt erläutert werden. Nicht alle Parameter lassen sich direkt messen. Viele hängen voneinander ab und müssen indirekt berechnet werden. Direkt messen lassen sich eigentlich nur die Größe der Platte und die Start- und Endzeitpunkte der Bearbeitung von einzelnen Aufträgen.

Bei fast allen Algorithmen wird ausgenutzt, dass die Zeitdauer zwischen dem Zeitpunkt an dem der Zielsektor gerade komplett unter dem Schreib-/Lesekopf entlang rotiert ist, bis zu dem Zeitpunkt, an dem der Festplatteninterrupt ausgelöst wird, relativ kurz ist und vor allem wenig Schwankungen unterliegt.

[6] Das TSC-Register als Zeitbasis des Moduls läuft auf einem 1 GHz Rechner etwa alle 585 Jahre über ($\frac{2^{64}}{1.000.000.000\ \mathrm{Hz} * 3.600\ \mathrm{s/h} * 24\ \mathrm{h/d} * 365\ \mathrm{d/a}} = 584,94\,\mathrm{a}$) und sollte damit keine Überlauf-Sonderbehandlung benötigen.

4.2.1 $time_{rotation}$

Das Prinzip zur Ermittlung der Rotationszeit ist relativ einfach und wird z. B. von Aboutabl et al. in [AAD97] erwähnt. Dieser Parameter hängt nicht von anderen ab.

Man muss dafür eigentlich nur zwei Aufträge an die Platte schicken, alles weitere dient der Erhöhung der Ergebnisgenauigkeit. Bei beiden Teilaufträgen greift man auf den selben Sektor zu. Von diesen muss jeweils der Endzeitpunkt aufgezeichnet werden. Die Zeitdifferenz dazwischen ist die Rotationsdauer. Zu beiden Endzeitpunkten befindet sich der Schreib-/Lesekopf ungefähr an der selben Stelle des Mediums. Aber nach dem zweiten Auftrag ist genau eine Rotationszeit vergangen.

Der zweite Auftrag sollte möglichst sofort nach dem Ende des ersten Auftrages abgesetzt werden, um nicht evtl. eine gesamte Rotation zu verpassen. Durch Mediumsfehler oder Verdrängung des messenden Prozesses direkt nach dem Ende des ersten Auftrages kann es zu weiteren Rotationen kommen.

Diese Messung kann sehr oft durchgeführt werden um den Fehlereinfluss durch zusätzliche Rotationen und verfahrensbedingte Messungenauigkeiten herauszumitteln. Durch Berechnung des arithmetisch getrimmten Mittels[7] im Gegensatz zum normalen arithmetischen Mittels ließ sich die Genauigkeit (verglichen mit der Herstellerangabe) und die Stabilität (über mehrer Durchläufe hinweg) stark verbessern. Dieses Verfahren bietet sich hier an, da dadurch eine bekannte und zu erwartende Fehlerursache eliminiert wird.

Um nicht durch Besonderheiten bestimmter Spuren oder Zonen die Messung zu verfälschen wird für jedes Zugriffspaar ein zufälliger Sektor auf der Platte ausgewählt.

```
// Daten aufnehmen
for (i = 0; i < DURCHLÄUFE; i++)
{
   sec = random(0 .. disc.size)
   time[i] = do_requests(disc, sec, sec)
}

// und nachbearbeiten
sort(time)
remove_upper_and_lower_10%(time)

// Ergebnis
double rotation_time = average(time);
```

4.2.2 Spurgrenzen

Um die Spurgrenzen zu identifizieren wird das zeitliche Verhalten von Aufträgen über Spurgrenzen hinweg ausgenutzt, welches durch den Spurversatz (siehe auch Abschnitt 2.1.3) beeinflusst wird. Dazu wird ein Zugriffspaar benutzt, welches jeweils einen Sektor schreibt und zwar n und $n + 1$.

Gemessen wird die Zeitdauer vom Ende des ersten bis zum Ende des zweiten Teilauftrages. Es ist zu erwarten, dass sich dieser Wert etwa im Rahmen der Plattenrotationszeit bewegt,

[7] Im Gegensatz zum arithmetischen Mittel werden z. B. die oberen und unteren 10% der Messwerte als Ausreißer eliminiert. Sie gehen nicht in die Berechnung ein.

da der zweite Zugriff fast bei dem selben Winkel endet wie der erste. Wenn aber der Sektor $n+1$ physisch nicht direkt nach dem Sektor n folgt, sondern er der erste Sektor der neuen Spur ist, so liegt zwischen ihnen ein deutlich größerer Winkel. Dieser Winkel schlägt sich auch in der gemessenen Zeit nieder. Mit diesem Verfahren wird nun für alle in Frage kommenden Auftragspaare die Ausführungszeit gemessen.

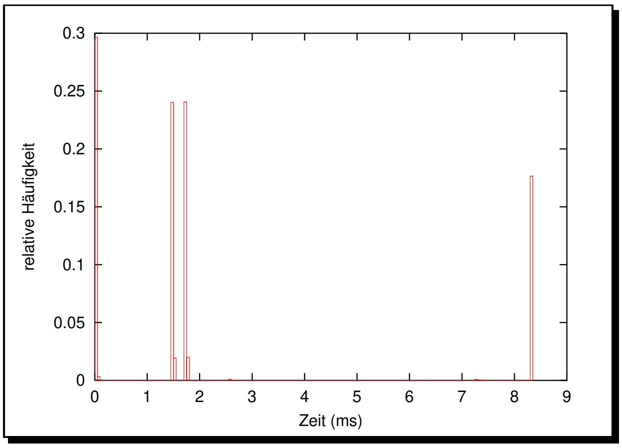


Abbildung 4.2: Bearbeitungszeiten für Auftragspaare mit den Zielsektoren $(n, n+1)$

Wenn man diese Zeiten modulo der Plattenrotationszeit (im Beispiel 8,3 ms) in ein Histogramm wie in Abbildung 4.2 einträgt, ergeben sich zwei große Häufungen. Die Anteile um 0 ms, bzw. 8,3 ms werden durch die normalen Zugriffe gebildet, bei denen der zweite Sektor direkt nach dem ersten auf der Platte liegt. Die Ausführungsdauer des Zugriffpaars beträgt hier ungefähr eine volle Rotation.

Die zweite Häufung der Ausführungszeiten bei etwa 1,5–1,8 ms wird durch Zugriffe über Spurgrenzen hinweg erzeugt. Je nach Festplattenserie lässt sich die zweite Häufung mehr oder weniger gut in zwei Teile trennen (im Beispielhistogramm vollständig möglich, bei etwa 1,6 ms), wobei der eine Teil vermutlich durch den Zylinderversatz, der andere durch den Kopfversatz verursacht wurde. Wie in Abschnitt 2.1.3 genauer erläutert, sollten beide etwa im selben Zeitbereich liegen. Eine Unterscheidung ist für die Anwendung in der vorliegenden Arbeit nicht wichtig, da durch beide Zeiten eine Spurgrenze gekennzeichnet wird.

Die Benutzung eines Zeitfensters mit der Größe der Rotationszeit der Platte schließt auch die Fehlerursache der mehrfachen Rotationen pro Zugriff aus. Einzelne Zugriffe brauchen einige Rotationen mehr, z. B. wegen falscher Positionierung der Köpfe oder Schreib-/Lesefehlern. Durch das Zeitfenster wird diese Fehlerquelle kompensiert.

Die Häufigkeiten in der Abbildung entsprechen nicht den echten Häufigkeiten auf der Platte, wo eine Spurgrenze z. B. nur alle 800 Sektoren vorkommt. Zur besseren Illustration wurden 100.000 zufällige Zugriffspaare und Zugriffspaare auf alle erkannten Spurgrenzen (108.894 bei dieser Platte) zusammengesetzt.

Gegenüber dem einfachen Ansatz, alle Nachbarpaare der Platte zu testen und zu entscheiden ob es sich um Spurgrenzen handelt oder nicht, wurde der Algorithmus stark optimiert. Durch Ausnutzung der Tatsache, dass sich die Spurgrößen zweier benachbarter Spuren fast nie, und wenn doch, dann nur leicht, unterscheiden, kann man sehr oft die nächste Spurgrenze erraten. Bei einem Fehlschlag des Rateversuches wird in der näheren Umgebung der geratenen Grenze weitergesucht. Trotz der Optimierungen dauert der Durchlauf dieser Messung bei einer 40 GB Platte mit 7.200 u/min ca. eine Stunde. Ohne die Optimierungen ist mit einem Zeitfaktor von 500 bis 800 zu rechnen.

Es gibt, außer den Spurgrenzen, noch eine andere Ursache für horizontale Verschiebungen im Histogramm — die Ausblendung fehlerhafter Sektoren. Sie könnten unter Umständen auch als Spurgrenze interpretiert werden, denn auch hier tritt zwischen zwei logisch benachbarten Sektoren ein deutlicher Versatz auf. Solche Stellen auf der Platte stellen praktisch Lücken im linearen Mapping dar.

4.2.3 $time_{max-skew}$

Dieser Parameter wird bei der Bestimmung der Spurgrenzen (siehe Abschnitt 4.2.2) mit ermittelt, da dort alle Spur-zu-Spur Zugriffszeiten anfallen. Das dort aufgetretene Maximum wird als $time_{max-skew}$ betrachtet. Der Wert des Parameters kann dabei durchaus weit von den typischen Werten (siehe Abbildung 4.2) abweichen. Das lässt sich mit einer Reihe von Faktoren begründen:

- ausgeblendeten Spuren (siehe auch Abschnitt 4.2.2),
- Unregelmäßigkeiten im Spurversatz und
- ausgeblendeten Sektoren, die am Ende oder Anfang der Spur liegen und deren Sektorrotationszeit praktisch zum Spurversatz zu addieren ist.

Abbildung 4.3 zeigt die Verteilung der Spurversätze auf der Seagate Barracuda ATA IV. Zur Ermittlung der Daten für die Grafik wurden Auftragspaare benutzt, die die beiden Sektoren um eine Spurgrenze einzeln beschreiben. Die Grafik stellt die Verteilung der Bearbeitungszeiten dar. Dabei wurde vom Ende des ersten zum Ende des zweiten Teilauftrages gemessen.

Die für den Worst-Case entscheidende Ausführungszeit liegt in der Abbildung bei etwa 2,8 ms.

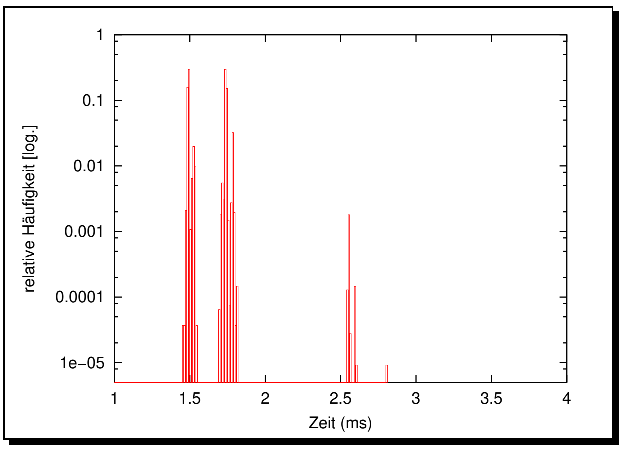


Abbildung 4.3: Spurversatzzeiten auf der Seagate Barracuda ATA IV

4.2.4 Zonen

Der Algorithmus zur Zonenerkennung führt keine Zugriffe auf die Platte aus. Er nutzt lediglich die gewonnenen Spurgrenzen (siehe auch Abschnitt 4.2.2). Zu jeder Spur ist auch ihre Größe in Sektoren bekannt. Der Algorithmus versucht nun in der Liste der Spuren Bereiche gleicher Größen zu erkennen. Durch transparente Ausblendung von fehlerhaften Sektoren gibt es aber in vielen Zonen Bereiche mit kleineren Spuren. Eine perfekte Zuordnung gelingt so manchmal nicht. Es entstehen einige kleine Zonen mit weniger Sektoren pro Spur inmitten großer homogener Zonen. Für die Anwendung in dieser Arbeit bereitet das aber keine großen Probleme, da die Zonen nicht von zentraler Bedeutung für die Bestimmung der Worst-Case-Zeiten sind. Sie werden eigentlich nur für die $time_{sector-rotation}$ (siehe Abschnitt 4.2.5) benötigt, die in jeder Zone zu ermitteln ist. Mit der Annahme, dass lineares Mapping vorliegt (was bei den Platten die untersucht wurden zutrifft) ließe sich die Erkennungsrate der Zonen wahrscheinlich noch weiter steigern.

4.2.5 $time_{sector-rotation}$

Dieser Parameter hängt von der Zone ab, in der die Zielspur liegt. Zur Berechnung müssen also die Zonen- und Spurgrenzen bekannt sein.

Die Idee hier ist ähnlich wie in Abschnitt 4.2.1: Es wird wieder ein Auftragspaar benutzt, das jeweils den selben Zielsektor hat. Beim zweiten Auftrag werden aber zwei Sektoren geschrieben. Gemessen wird vom Ende des ersten bis zum Ende des zweiten Auftrages. Der zweite Auftrag endet damit genau einen Sektor später als der erste. Die gemessene Zeitdauer ist also die Summe aus einer vollen Rotationsdauer (siehe Abschnitt 4.2.1) und der Zeit, die benötigt wird um einen weiteren Sektor vollständig unter dem Kopf hindurch zu rotieren:

$$time_{request} = time_{rotation} + time_{sector-rotation} \tag{4.1}$$
$$time_{sector-rotation} = time_{request} - time_{rotation} \tag{4.2}$$

Seek-Zeiten fallen nicht an, da beide Aufträge in die selbe Spur zielen. Dadurch, dass der Kommando-Overhead für beide Aufträge als gleich groß vorausgesetzt wird, verzögern sich auch die beiden Endzeitpunkte der Teilaufträge gleich und beeinflussen die zu messende Zeitdauer dazwischen nicht.

Bei der Implementierung wurde nicht nur ein Sektor zusätzlich geschrieben, sondern eine möglichst große Anzahl. Diese Anzahl muss vollständig in die Zielspur passen, damit kein Spurversatz das Ergebnis verfälscht. Außerdem muss die Anzahl in einem Auftrag zu bewältigen sein, damit keine Zusatzrotationen auftreten. Durch die beschriebene Vorgehensweise wird die Messgenauigkeit erhöht.

Diese Messung wird für jede ermittelte Zone der Platte durchgeführt.

4.2.6 $time_{overhead}$

Wenn man hintereinander zwei Aufträge an die Platte schickt, die auf Sektor n und Sektor $n + 1$ zugreifen, so wird man in der Regel mehr als eine volle Rotation warten müssen, da zu dem Zeitpunkt, zu dem die Platte den zweiten Auftrag ausführen will, der Sektor $n + 1$ den Kopf schon passiert hat. Wenn man nun schrittweise den Zielsektor des zweiten

Auftrages nach hinten verschiebt $(n+m)$, so wird die Bearbeitungszeit des Auftragspaares mit steigendem m ansteigen, bis sie plötzlich wieder fällt und zwar deutlich unter die Rotationszeit der Platte. Bei diesem Abstand zwischen den zwei Sektoren entspricht die Rotationszeit zwischen ihnen dem Parameter $time_{overhead}$. Mit Hilfe der in Abschnitt 4.2.5 ermittelten $time_{sector-rotation}$ kann $time_{overhead}$ nun berechnet werden:

$$time_{overhead} = (m-1) * time_{sector-rotation} \tag{4.3}$$

Für $m = 1$ wäre $time_{overhead} = 0$, da zwischen dem Ende des Zielsektor des ersten Auftrages (n) und dem Beginn des zweiten $(n+1)$ kein Abstand auf der Platte ist.

Diese Messung sollte möglichst in der schnellsten Plattenzone stattfinden, da die zeitliche Auflösung der Messung an die Größe von $time_{sector-rotation}$ gebunden ist und dieser Parameter dort am kleinsten ist.

4.2.7 $time_{seek}$

Zum Ermitteln der Worst-Case-Seek-Zeit werden Auftragspaare benutzt, die den Kamm von einem Ende der Platte zum anderen bewegen. Die bei Versuchen praktisch gemessenen Zeiten sind Summen von Teilzeiten wie in Abschnitt 3.2.2 erläutert. Durch Ermittlung der anderen Summanden lässt sich $time_{seek}$ berechnen:

$$time_{seek} = time_{request} - time_{request-rotation} - time_{sector-rotation} - time_{overhead} \tag{4.4}$$

- $time_{request}$ ist hier die Dauer des ausgeführten Plattenauftrages. Dieser Parameter wird also direkt gemessen.
- $time_{request-rotation}$ ist der Anteil der Gesamtzeit, der auf die Rotation der Platte gewartet wird. Dieser Zeitraum beginnt nach dem erfolgreichen positionieren des Schreib-/Lesekopfes in der Zielspur. Dieser Summand kann vernachlässigt werden, wenn der Zielsektor so gewählt wird, dass direkt nach dem der Seek abgeschlossen ist, der Zielsektor unter dem Kopf beginnt. Das wird erreicht, indem als Zielsektor jeder Sektor in der Zielspur getestet wird. Bei der minimalen Gesamtausführungszeit ist der gewünschte Sektor erreicht. Dieser Teilalgorithmus wird durch einige Wiederholungen robuster gegenüber Schwankungen in der $time_{seek}$. Gleichzeitig kann durch Intervallschachtelung und zweistufiges Vorgehen (erst grob, dann fein) beim Suchen des richtigen Zielsektors wieder viel Zeit gespart werden.
- $time_{sector-rotation}$ hängt von der Zone des Zielsektors ab. Der Algorithmus zur Ermittlung wird in Abschnitt 4.2.5 näher beschrieben.
- Wie $time_{overhead}$ bestimmt wird ist genauer in Abschnitt 4.2.6 erläutert.

Die in diesem Modell benutzte $time_{seek}$ besteht aus zwei Komponenten, der Seek-Zeit an sich und der Kopfumschaltzeit. Wenn die Start- und die Zielspur vom selben Kopf bearbeitet werden (siehe Abschnitt 2.1.6) entfällt diese Zeit. In der Implementierung wird deswegen nicht einfach nur der Sprung von der ersten zur letzten Spur gemessen, sondern es werden viele Sprünge von einigen Spuren am Plattenanfang auf einige Spuren am Plattenende durchgeführt. Von diesen wird die Maximalzeit berechnet. Dadurch wird die maximale Kopfumschaltzeit in $time_{seek}$ berücksichtigt.

4.2.8 Seek-Kurve

Für die Bestimmung der Worst-Case-Zeit eines Plattenauftrages hat die Seek-Kurve eigentlich nur einen Zweck: aus ihr kann man erkennen, bei welchen Zugriffen die längsten Seek-Zeiten entstehen. In der vorliegenden Arbeit wird die Seek-Kurve benutzt, um zu überprüfen ob auf den zu vermessenden Platten lineares Mapping benutzt wird, d. h. ob bei Zugriffen vom logischen Anfang der Platte zum logischen Ende auch Seeks mit der maximalen Antwortzeit erzeugt werden.

4.2.9 Verteilung der Zusatzrotationen — n

Dieser Algorithmus besteht aus zwei Teilen und soll jeweils die Verteilung und damit auch das Maximum des Parameters n (siehe Abschnitt 3.2.3 ff.) ermitteln:

1. Der erste Teil bezieht sich auf das Modell aus Abschnitt 3.2.3. Hier wird die Verteilung des Parameters n durch praktische Messungen mit Zugriffen auf *einen* Sektor ermittelt. Als Grenzwert für $n = 1$ wird dabei die Zeitgrenze aus Modell 3.1 benutzt. Damit ein Zugriff mit $n = 2$ eingestuft wird, muss er eine Umdrehung länger brauchen, für $n = 3$ zwei Umdrehungen usw..

2. Der zweite Teil bezieht sich auf das erweiterte Modell aus Abschnitt 3.2.4. Dabei werden Zugriffe mit *mehr als einem* Sektor durchgeführt. Der Grenzwert für $n = 1$ wird hier durch das Modell 3.4 mit $n = 1$ bestimmt. Für die Einstufung eines Auftrages mit $n = 2, 3, \ldots$ sind wieder entsprechend viele Zusatzumdrehungen der Platte notwendig. Bei diesem Teil des Algorithmus verändert sich der Grenzwert mit der Größe des Plattenauftrages, da die Größe in Modell 3.4 als Parameter eingeht.

Diese zwei Versuche müssen mit einer großen Anzahl von Wiederholungen durchgeführt werden, um repräsentative Aussagen zu erhalten.

4.2.10 Anzahl der Spurversätze pro Auftrag — v

Bei den betrachteten Platten lag die Anzahl der Sektoren pro Spur im Bereich von 350 bis 850 (je nach Zone). Mit der weiteren Entwicklung in diesem Bereich ist eher mit mehr Sektoren pro Spur zu rechnen, da die Datendichte auf den Scheiben weiter gesteigert wird. Die aktuelle Implementierung ist auf IDE beschränkt und der IDE-Plattentreiber von Linux erzwingt, wie in Abschnitt 3.2.4 erwähnt, die Limitierung auf 255 Sektoren pro Auftrag.

Aus diesen Gründen wird der Parameter v bei Aufträgen mit mehr als einem Sektor gleich eins gesetzt, im anderen Fall gleich null.

4.3 Verteilung und Zugriffsmuster

Um die Verteilung der Bearbeitungszeiten von Plattenaufträgen messen zu können kann die gleiche Methode wie in Abschnitt 4.1.4 benutzt werden. Die Aufzeichnung der Muster erfolgt zeitgleich zum Durchlauf der Zielanwendung mit Hilfe der Schnittstelle für die Zeitmessung. Die aufgezeichnete Zugriffsfolge wird dabei von einer Reihe Faktoren beeinflusst:

- Der Puffercache von Linux kann einen Teil der lesenden Plattenaufträge der Anwendung selber beantworten, da er die entsprechenden Blöcke zwischengespeichert hat. Hier spielt vor allem die Größe des zu Verfügung stehenden Hauptspeichers und der aktuelle Füllstand des Puffercaches eine Rolle.
- Der Puffercache kann schreibende Aufträge verzögert an die Platte weiterreichen.
- Der Puffercache kann bei mehrfachem Zugriff auf die selben Sektoren weniger Schreibzugriffe an die Platte durchreichen.
- Die Read-Ahead-Strategie kann bestimmte Sektoren von der Platte holen, auf die die Anwendung noch gar nicht zugreifen will. Dazu werden auch kleinere Aufträge in größere umgesetzt.
- Der Platten-Scheduler kann die Ausführungsreihenfolge der Aufträge verändern und mehrere benachbarte kleine zu größeren Aufträgen zusammenfassen.
- Zusätzliche Zugriffe auf die Metadaten des Dateisystems sind notwendig.

Die Verteilung der Ausführungszeiten der tatsächlich ausgeführten Aufträge wird von weiteren Gegebenheiten beeinflusst:

- der Größe und dem Füllstand des Plattencaches,
- dem allgemeinen Zeitverhalten der Platte und der Anbindung (Rotationszeit, Seek-Geschwindigkeit, Bus, Übertragungsmodus),
- der Zone der Platte, in der die Messung stattfindet und
- der allgemeinen Geschwindigkeit des Rechners.

Aufgrund dieser vielen Abhängigkeiten erschien die Aufzeichnung der vollständigen Zugriffsfolge als der sinnvollste Weg der Implementierung. Die Folge besteht aus einer einfachen Liste der Zugriffe, wobei jedes Listenelement den Startsektor, die Größe des Plattenauftrages und die Art des Zugriffes (schreibend oder lesend) enthält. Die Zeitdauer zwischen zwei Aufträgen (d. h. zwischen Ende des ersten und Beginn des zweiten) wird vernachlässigt (siehe auch Abschnitt 3.3). Mit diesen Daten kann man die Folge leicht wieder auf eine Platte abspielen. Dabei können z. B. folgende Informationen erfasst werden:

- Verteilung der Bearbeitungszeiten,
- Anteil der schreibenden und Anteil der lesenden Zugriffe,
- Größenverteilung der Aufträge und
- Verteilung der Seek-Abstände.

Die Dateien mit den vollständigen Zugriffsmustern können relativ groß werden. Um Seiteneffekte auf die Messung selbst zu vermeiden wurden sie meist in eine Ramdisk ausgegeben. Denkbar wäre auch, sie auf eine Platte an einem anderen Bus zu schreiben, bzw. sie beim Abspielen von dort zu lesen.

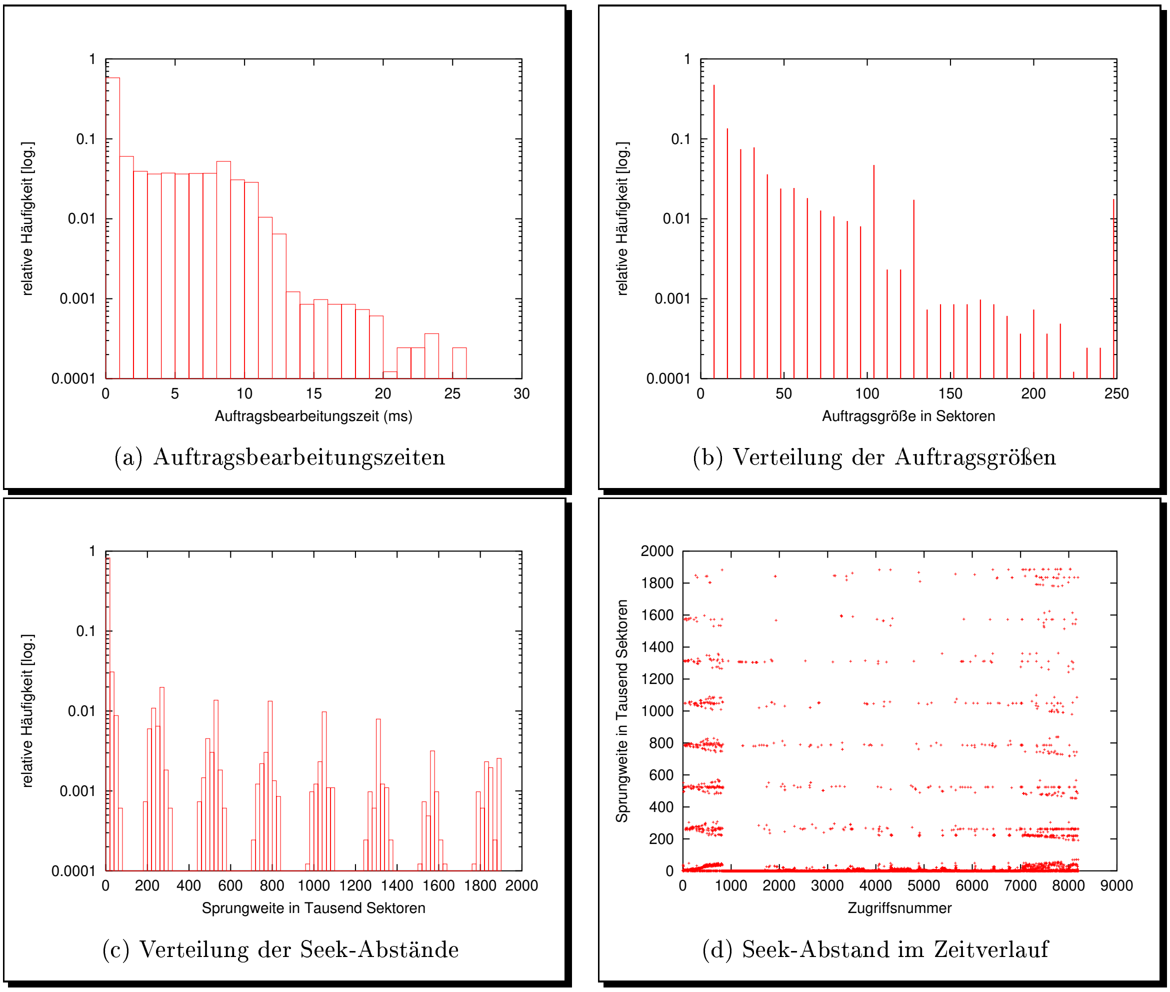
4.3.1 Ein Beispiel

Als Beispiel soll hier die Durchführung der Aufnahme einer vollständigen Kernkompilierung erläutert werden. Dazu wurden folgende Schritte ausgeführt:

1. Aufnahme vorbereiten
    a) Platte partitionieren (`fdisk /dev/hdd ...`)
    b) Dateisystem anlegen und mounten (`mke2fs /dev/hdd1; mount /dev/hdd1 /mnt/tmp`)
    c) Kernquellen kopieren, Konfiguration festlegen (`make menuconfig`), aufräumen (`make clean`)
2. Aufnahme des Musters durchführen
    a) Neustart des Rechners um Cache von Linux zu leeren
    b) Umgebung vorbereiten (Schreibcache der Zielplatte deaktivieren, Ramdisk für Ausgabedaten anlegen)
    c) Zielpartition mounten (`mount /dev/hdd1 /mnt/tmp`)
    d) Aufnahme beginnen
    (`average -t /dev/hd_perf4 -s 1000.0 > ramdisk/kernel-make.rec`)
    e) aufzunehmende Anwendung starten (hier: `make clean dep bzImage modules`)
    f) Aufnahme beenden (`ctrl-c`) und Ergebnis aus der Ramdisk sichern
3. Aufnahme abspielen und Zeitverhalten vermessen
    a) zu vermessende Platte als *raw-device* kapseln (`raw /dev/raw1 /dev/hdd`)
    b) Aufnahme erneut starten, dieses Mal um die Bearbeitungszeiten zu messen (`average -t /dev/hd_perf4 -s 1000.0 > ramdisk/timing.rec`)
    c) Aufgenommenes Muster abspielen (`replay -r kernel-make.rec /dev/raw1`)
    Achtung: Es ist sehr wahrscheinlich, dass das Dateisystem und die Partitionstabelle, welche auf der in `raw1` gekapselten Platte liegen, zerstört werden!
    d) nach dem Ende des Abspielvorganges die Aufnahme beenden (`ctrl-c`)
    e) Logdatei mit Zeitinformationen aus der Ramdisk sichern und analysieren ...

Schritt 3 kann jetzt mit jeder beliebigen Platte wiederholt werden. Aus der Analyse der Logdatei lassen sich z. B. folgende Informationen gewinnen:

- die Verteilung der Auftragsbearbeitungszeiten (Abbildung 4.4(a)),
- die Verteilung der Auftragsgrößen (Abbildung 4.4(b)),
- die Verteilung der Seek-Abstände (Abbildung 4.4(c)) und
- das zeitliche Verhalten der Seek-Abstände (Abbildung 4.4(d)).



(a) Auftragsbearbeitungszeiten

(b) Verteilung der Auftragsgrößen

(c) Verteilung der Seek-Abstände

(d) Seek-Abstand im Zeitverlauf

Abbildung 4.4: Ergebnisübersicht des „kernel-make"-Musters auf der IBM Deskstar 40 GB

Tabelle 4.1: Sonstige Daten des Zugriffsmusters

Plattenaufträge:	8203
Anteil Lese-/Schreibaufträge:	83,1% / 16,9%
Durchschnittliche Bearbeitungszeit:	2,69 ms
Durchschnittliche Auftragsgröße:	32,2 Sektoren
Durchschnittlicher Seek-Abstand:	103179 Sektoren

Die Graphen in Abbildung 4.4 wurden bei einer Vermessung der IBM Deskstar 40 GB mit dem oben beschriebenen Muster gewonnen, ebenso die Daten in Tabelle 4.1. Charakteristisch für die Platte sind dabei nur die Zeitwerte (a), die anderen Daten werden vollständig durch das Zugriffsmuster bestimmt.

Interessant ist zum Beispiel, dass nur ganzzahlige Vielfache von acht als Auftragsgröße auftreten (b). Das lückenhafte Histogramm der Seek-Abstände (c) rührt wahrscheinlich von der Anordnung der Dateien im Dateisystem (hier *Ext2*) her. Die große Häufung um 0 entsteht durch den Anteil der sequentiellen Zugriffe im Muster. Diese Zugriffe können — insofern sie lesend erfolgen — größtenteils durch das Prefetching der Platte befriedigt werden und bilden deswegen in (a) auch eine starke Häufung um 0 ms. Der Schreibcache der Platte war in diesem Versuch ausgeschaltet.

5 Leistungsbewertung

Praktische Messergebnisse der Worst-Case-Zeitmessung sind in Abschnitt A für zwei Festplatten dargestellt. Die Aufnahme, das Abspielen und das Vermessen von Plattenzugriffsmustern wurde in Abschnitt 4.3.1 demonstriert. Für beide Teilziele wurde eine funktionierende Implementierung geschaffen.

Der Kernpatch mit dem Modul für die Zeitmessung liefert extrem genaue Ergebnisse, die deutlich über die Anforderungen hinausgehen, den Rechner in seiner Leistungsfähigkeit aber praktisch nicht beeinträchtigt. Gleichzeitig liefert diese Lösung auch einen transparenten Weg um Plattenzugriffe an sich mitzuprotokollieren, z. B. die Blocknummer und die Größe des Zugriffes.

5.1 Stand der Implementierung

Die Implementierung des Werkzeuges für die Messung der Worst-Case-Zeiten („`worst_case`”) ist für die gestellten Ziele einsatzfähig und liefert sehr gute Resultate. Für jede Teilkomponente der benutzen Modelle (siehe auch Abschnitt 3.2) konnte eine Messmethode entwickelt und implementiert werden. Der Aufbau des Programms ist sehr modular und erlaubt die Wiederverwendung einzelner Mechanismen auch in anderen Programmen. Insbesondere wurde eine Sammlung von Algorithmen implementiert, die sich mit einzelne Parameter beschäftigen. Beispielsweise wäre hier die Ermittlung der Rotationsdauer der Platte zu nennen. Eine Erweiterung um neue Teile ist leicht möglich, sollten neue Anforderungen sichtbar werden. Für grundlegende Probleme, wie z. B. das Absetzen von Zugriffspaaren und die Messung der Ausführungsdauer, wurde eine Sammlung von Funktionen zusammengestellt.

Teile dieser Bibliotheken konnten auch bei den Werkzeugen für die Ermittlung der Zugriffsmuster benutzt werden. Für diese Teilaufgabe wurden zwei Programme geschaffen, eines zum Aufnehmen der Zugriffsmuster und der Bearbeitungszeiten („`average`”) und eines um die Muster wieder abzuspielen („`replay`”). Es hat sich gezeigt, dass für die Aufnahme der Muster und der Bearbeitungszeiten gleich vorgegangen werden kann, da mithilfe des Zeitmessungsmoduls immer alle Informationen für die Einzelzugriffe erfasst werden.

Der Kernpatch und das Modul für die Zeitmessung liefen im gesamten Einsatzzeitraum stabil. Insofern erscheint es möglich dieses Modul auch längerfristig auf Produktivrechnern einzusetzen, um praktisch relevante Zugriffsmuster aufzuzeichnen.

Für die automatische Vermessung der Platten wurden Skripte geschrieben, die notwendige Voreinstellungen vornehmen, die einzelnen Programme mit entsprechenden Parametern aufrufen und aus den gemessenen Werten gewünschte Daten extrahieren und Graphen erstellen. Dazu wurden kleinere Programme implementiert, die z. B. Histogramme erzeugen oder Mittelwerte u. ä. aus Datenmengen berechnen. Für die Erstellung der Graphen wurde auf „`gnuplot`” zurückgegriffen.

Das ganze Werkzeugpaket konnte bisher nur an zwei Festplatten vollständig ausprobiert werden. Während der Implementierung wurden einzelne Algorithmen aber auch mit weiteren Platten überprüft. Eventuell werden noch Anpassungen beim Einsatz mit anderen Plattenserien notwendig.

6 Zusammenfassung und Ausblick

Die vorliegende Arbeit beschreibt den Entwurf und die Implementierung einer Werkzeugsammlung für die Ermittlung der Laufzeiteigenschaften von Festplatten. Es wird eine Lösung für die Ermittlung der Worst-Case-Zeiten und eine für die Ermittlung der Verteilungen von Bearbeitungszeiten bestimmter Zugriffsmustern gezeigt. Des Weiteren wird demonstriert, wie Zugriffsmuster aufzunehmen und abzuspielen sind. Die Werkzeuge die als Ergebnis dieser Arbeit entstanden, sind einsatzfähig und liefern praktisch relevante Daten.

Für zukünftige Arbeiten bieten sich eine Reihe von Anknüpfungspunkten an:

- Für den Einsatz der Werkzeuge mit moderneren Platten und für eine SCSI Portierung müssten die vorgestellten Plattenmodelle einige Eigenschaften solcher Platten mit berücksichtigen. Das wären z. B. *Zero-Latency-Access* (auch bekannt als *Read-/Write-on-arrival*) und *Command-Queueing*.

- Der Ansatz Plattenzugriffsmuster 1:1 aufzunehmen und auf andere Platten abzuspielen bringt beim Übergang auf größerer Platten Probleme, da hier immer nur Teile der neuen Platten abgedeckt werden. Mögliche Ansatzpunkte wären hier:

  - eine Spreizung der Zugriffsmuster auf die ganze Platte,
  - das Abspielen der Muster in verschiedenen Zonen der großen Platte und
  - Charakterisierung der Muster und Generierung neuer Muster, die der Charakterisierung entsprechen, aber gleichzeitig den größeren Bereich der neuen Platte abdecken.

- Das Ausgabeformat der Werkzeuge könnte auf XML umgestellt werden, damit auch die Datenbanken verschiedener Versionen der Werkzeuge benutzbar bleiben.

- Die ermittelten Daten, insbesondere das Mapping und die Seek-Kurven könnten in Platten-Schedulern benutzt werden um das Umordnen der Aufträge zu optimieren.

- Eine Portierung der Werkzeuge für SCSI-Platten kann sinnvoll sein. Dazu müssten evtl. die Modelle und einzelne Algorithmen etwas angepasst werden. Außerdem muss der Treiber des gewünschten SCSI-Adapters für die Zeitmessung leicht verändert werden.

- Eine statistische Untersuchung der Wiederholungsanzahl bestimmter Teilalgorithmen und ihrer Ergebnisse könnte Aufschluss über die Sicherheit der gewonnenen Zeitwerte liefern.

Anhang A

Messergebnisse

In diesem Abschnitt sollen einige praktische Messergebnisse der Arbeit vorgestellt werden. Alle hier gezeigten Daten wurden mit den für diese Arbeit entwickelten Werkzeugen aufgenommen. Graphen wurden mit „`gnuplot`" erzeugt. Histogramme wurden mit einer verbesserten Version eines kleinen Kommandozeilenprogramms von Mustafa Kocaturk erstellt.

Alle Messungen erfolgten auf einem Athlon 1000 MHz mit 512 MByte Hauptspeicher unter Linux (Kernversion 2.4.6, incl. dem Patch für die Zeitmessung).

A.1 Seagate Barracuda ATA IV

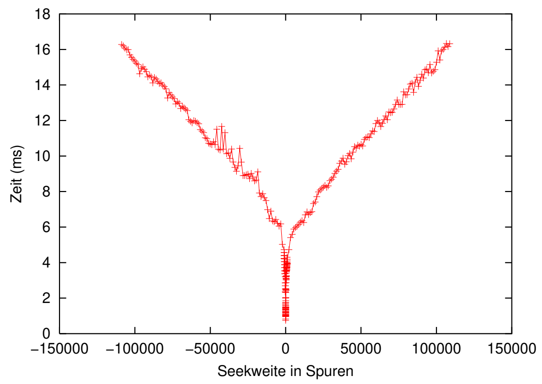
Tabelle A.1: Herstellerangaben

Hersteller:	Seagate
Modellbezeichnung:	Barracuda ATA IV
Modellnummer:	ST340016A
Seriennummer:	3HS1LG8S
Kapazität (LBA-Sektoren):	40 GB (78.165.360)
Cache-Größe:	2 MB
Sektorgröße:	512 Byte
Anschlusstyp:	ATA, UDMA-100
Rotationsgeschwindigkeit:	7200 min^{-1}

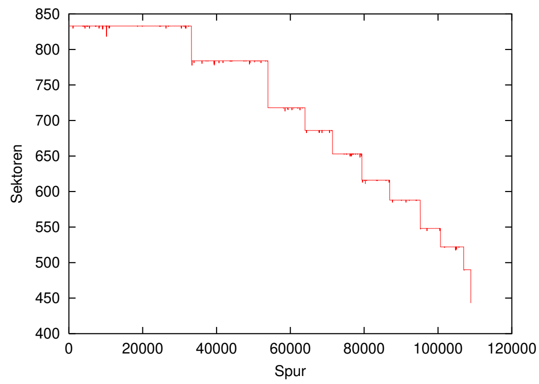
Tabelle A.2: Gemessene und berechnete Werte

Worst-Case-Seek-Zeit:	17,21 ms
Kommando-Overhead:	0,66 ms
Rotationszeit:	8,33 ms
Spuren:	108895
maximaler Spurversatz:	2,82 ms
Worst-Case für 1-Sektor Aufträge (laut Modell 3.2):	**42,88 ms**
Worst-Case für 128-Sektor Aufträge (laut Modell 3.4):	**47,84 ms**

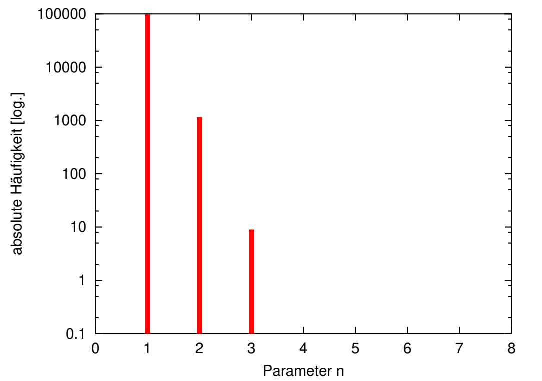
Besonderheiten: Die Seagte-Platte hat im Vergleich zur IBM Deskstar 40 GB einen deutlich weniger verrauschten Seek-Graph. Die Platte scheint außerdem in wärmerer Umgebung Probleme bei Lesezugriffen zu haben. Nach längerem Betrieb mit geschlossenem Rechnergehäuse dauerten Worst-Case-Lesezugriffe bis zu **75 ms**, gegenüber ca. **45 ms** in kühlem Zustand. Schreibzugriffe waren davon nicht betroffen.



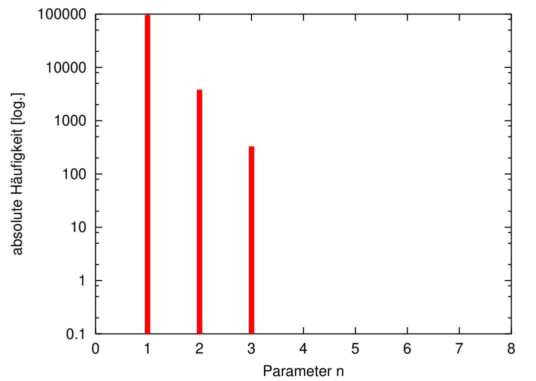
(a) Seek-Graph



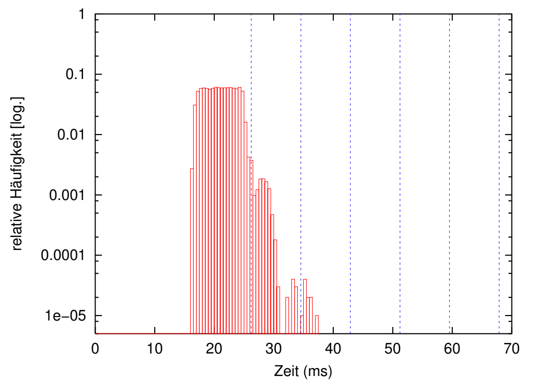
(b) Zonenaufteilung



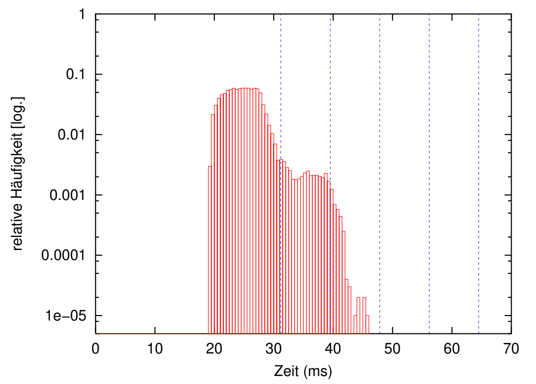
(c) Verteilung für n bei 1-Sektor-Zugriffen (full-stroke)



(d) Verteilung für n bei 128-Sektor-Zugriffen (full stroke)



(e) Verteilung der Bearbeitungszeiten bei 1-Sektor-Zugriffen (full stroke)



(f) Verteilung der Bearbeitungszeiten bei 128-Sektor-Zugriffen (full stroke)

Abbildung A.1: Übersicht über die Festplatte Seagate Barracuda ATA IV (ST340016A)

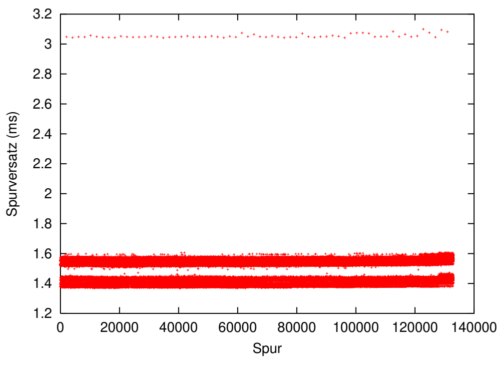
A.2 IBM Deskstar 40 GB

Tabelle A.3: Herstellerangaben

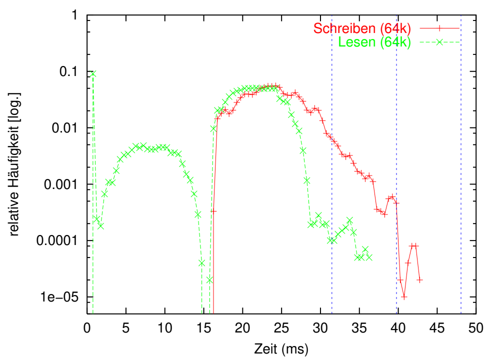
Hersteller:	IBM
Modellbezeichnung:	Deskstar 40 GB
Modellnummer:	IC35L040AVER07-0
Seriennummer:	SXPTXHT6796
Kapazität (LBA-Sektoren):	40 GB (80418240)
Cache-Größe:	2 MB
Sektorgröße:	512 Byte
Anschlusstyp:	ATA, UDMA-100
Rotationsgeschwindigkeit:	7200 min^{-1}

Tabelle A.4: Gemessene und berechnete Werte

Worst-Case-Seek-Zeit:	16,80 ms
Kommando-Overhead:	0,41 ms
Rotationszeit:	8,30 ms
Spuren:	133009
maximaler Spurversatz:	3,10 ms
Worst-Case für 1-Sektor Aufträge (laut Modell 3.2):	**33,83 ms**
Worst-Case für 128-Sektor Aufträge (laut Modell 3.4):	**48,07 ms**



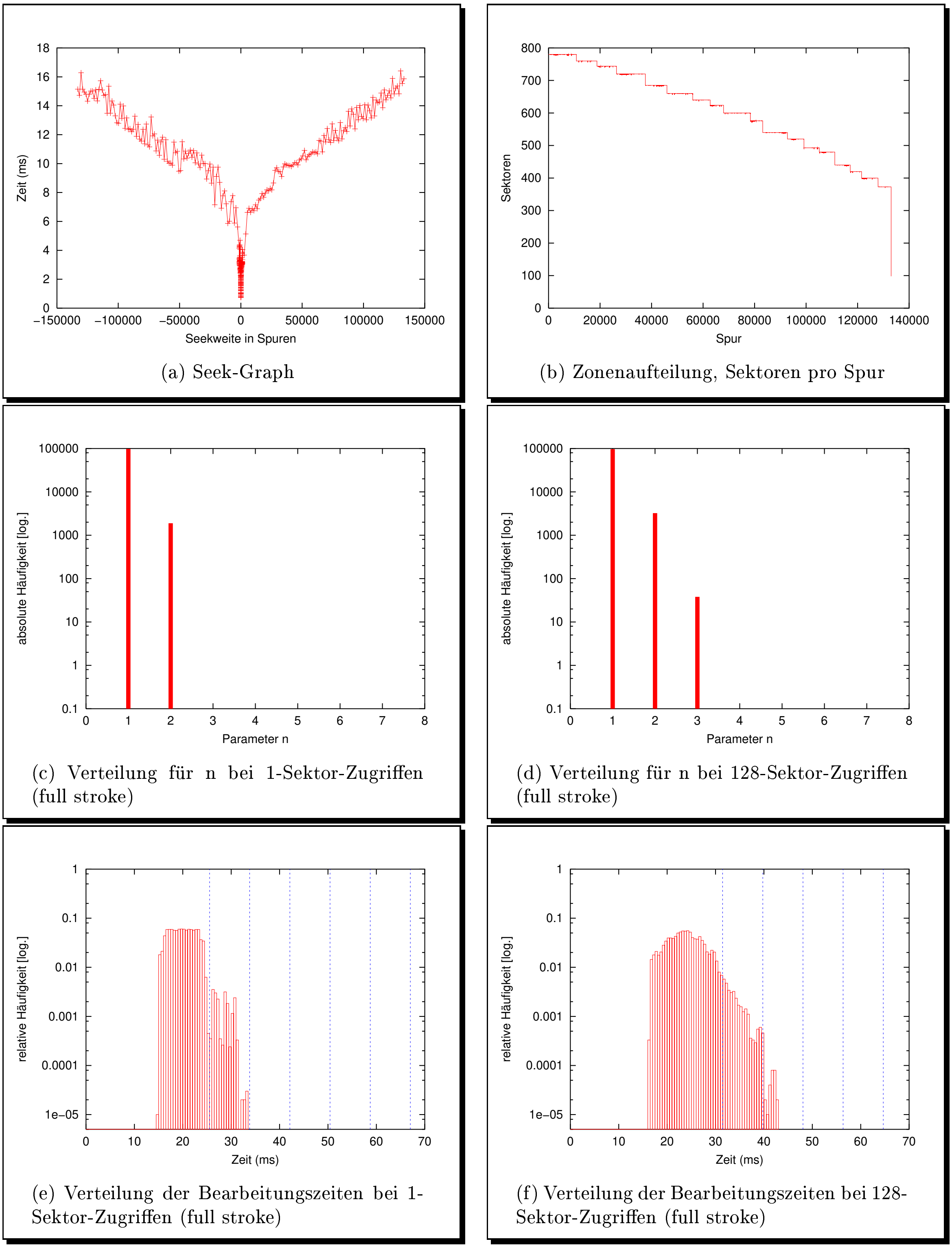
(a) Über die gesamte Platte verteilte Spurversatzzeiten



(b) Verteilung der Bearbeitungszeiten bei 128-Sektor-Zugriffen (full stroke)

Abbildung A.2: Besonderheiten der IBM Deskstar 40 GB

Besonderheiten: Bei der IBM-Platte fielen besonders die guten Worst-Case-Zeiten beim Lesen auf (siehe Abbildung A.2(b)). Der Spurversatz in Abbildung A.2(a) weist außerdem sehr regelmäßige Unterbrechungen mit der doppelten Versatzzeit auf, was auf ausgeblendete Reservespuren deutet. Der Seek-Graph ist relativ stark verrauscht.



(a) Seek-Graph

(b) Zonenaufteilung, Sektoren pro Spur

(c) Verteilung für n bei 1-Sektor-Zugriffen (full stroke)

(d) Verteilung für n bei 128-Sektor-Zugriffen (full stroke)

(e) Verteilung der Bearbeitungszeiten bei 1-Sektor-Zugriffen (full stroke)

(f) Verteilung der Bearbeitungszeiten bei 128-Sektor-Zugriffen (full stroke)

Abbildung A.3: Übersicht über Festplatte IBM Deskstar 40 GB (IC35L040AVER07-0)

Anhang B

Glossar

ATA (*AT attachment*) ist eine Anschlussschnittstelle für Speicherperipherie. Sie kommt heute vorwiegend in Desktop Rechner zum Einsatz.

Cache Caches werden allgemein eingesetzt um Daten an bestimmten Orten vorrätig zu halten. Ziel ist meist die Optimierung der schnellen Verfügbarkeit. Festplattencaches bestehen aus flüchtigem RAM und werden in zwei Situationen eingesetzt. Leseaufträge werden aus dem Cache beantwortet wenn die gewünschten Sektoren vorrätig sind. Der Zugriff auf den Cache geht deutlich schneller als der physische Zugriff auf den Datenträger. Schreibaufträge werden zuerst im Cache gepuffert um den Plattentreiber nicht lange zu blockieren. Sie werden irgendwann später von der Platte ausgeführt. Außerdem kann die Abarbeitungsreihenfolge verändert werden, sodass eine Auftragsfolge im Cache mit weniger Kammbewegungen auf die Platte geschrieben werden kann.

CHS ist ein älteres Verfahren zur Adressierung von Sektoren auf der Platte. Die Position des Sektors wird dabei durch ein Tripel, bestehend aus Zylinder, Kopf und Sektor beschrieben.

DMA (*direct memory access*) bezeichnet allgemein Zugriffe auf den Hauptspeicher eines Rechners ohne direkte Benutzung des Prozessors. Im Gegensatz zum **PIO-Modus** ist der Prozessor in der Zeit der Datenübertragung nicht blockiert.

Echtzeit bezeichnet allgemein, ob ein bestimmtes Ergebnis innerhalb definierter Zeitgrenzen vorliegt. Bei Festplatten ist die Bearbeitungszeit bestimmter Schreib- oder Lesezugriffe gemeint.

- **harte** Wenn ein Anfrageergebnis nicht innerhalb der zugesicherten Zeitgrenze vorliegt ist es wertlos. Der gesamte Prozess ist gescheitert.
- **weiche** Für weiche Echtzeit gibt es verschiedene Definitionen. Eine Möglichkeit ist, den Wertverlust für ein Teilergebnis nicht an eine harte Schranke zu binden, sondern eine von der Antwortzeit abhängige Wertfunktion zu definieren. Die harte Echtzeit ist also ein Spezialfall mit einer Wertfunktion, die nach einer Zeitschranke auf null fällt. Alternativ dazu existiert die Möglichkeit die harte Echtzeitschranke nur für einen bestimmten Anteil der Aufträge zu garantieren. Bei harter Echtzeit muss die Einhaltung der Zeitschranke für jeden einzelnen Auftrag gesichert sein.

IDE (integrated device electronics) siehe **ATA**

LBA Die Herkunft dieser Abkürzung scheint umstritten. Während es in [AAD97] als „*Logical Block Addressing*" bezeichnet wird, schreibt Köhntopp in [Köh96] „*linear block addresses*". Generell ist damit die Adressierung des gesamten Speicherbereiches der Festplatte über logische Blocknummern gemeint.

Mapping ist der Vorgang logische Blockadressen auf physische Sektoradressen abzubilden.

PIO (programmed I/O) bezeichnet ein Speicherzugriffsverfahren, bei dem der Prozessor jedes einzelne Datum zwischen Speicher und Ein-/Ausgabekanal transportiert. Der Prozessor ist hier mit dem Datenaustausch ausgelastet (im Gegensatz zu **DMA**).

Prefetching bezeichnet allgemein den Mechanismus bestimmte Daten, von denen vermutet wird, dass sie in Zukunft benötigt werden, in den Cache zu holen. Prefetching kann auf mehreren Ebenen geschehen: auf der Platte selbst in ihren Cache, vom Betriebssystem in den Hauptspeicher oder direkt von Anwendungsprogrammen.

Read-Ahead siehe **Prefetching**

SCSI (*small computer systems interface*) ist ein Anschlussbus für Speicher- und andere Peripherie. SCSI wird heute vorwiegend in Servern eingesetzt.

Sektor ist die kleinste Speichereinheit auf Datenträgern. Bei Festplatten sind heute 512 Byte pro Sektor verbreitet.

Settle / Resettle ist der Feinpositioniervorgang des Schreib-/Lesekopfes auf die Zielspur.

SMART (*Self-Monitoring, Analysis, and Reporting Technology*) ist eine Technik, die zur Laufzeit bestimmte Parameter von Festplatten überwacht und Abnutzungserscheinungen vor dem Ausfall Versagen erkennen kann.

Versatz (skew) bezeichnet allgemein den Abstand zwischen zwei Sektoren. Dieser kann als Winkel oder bei konstanter Rotationsgeschwindigkeit als Zeitdauer angegeben werden.

- **Zylinderversatz** ist der Versatz zwischen dem letzten Sektor einer Spur und dem ersten Sektor der nächsten, wobei beide Spuren in benachbarten Zylindern liegen und logisch aufeinander folgen.
- **Kopfversatz** ist der Versatz zwischen dem letzten Sektor einer Spur und dem ersten Sektor der nächsten, wobei zwischen den beiden Spuren nur eine Kopfumschaltung stattfindet, beide also im selben Zylinder liegen.
- **Spurversatz** ist hier Oberbegriff für Zylinder- und Kopfversatz.

Worst-Case (ungünstigster Fall) Wenn harte Echtzeitgarantien gegeben werden sollen, muss immer mit dem ungünstigster Fall gerechnet werden. Der Worst-Case bei Plattenaufträgen ist derjenige, der am längsten dauert.

Zoning / Zoned-Bit-Recording bezeichnet das Verfahren, in den äußeren Bereichen eines rotierenden Datenträgers pro Spur mehr Daten unterzubringen als in den inneren. Bei konstanter Drehzahl ergibt sich damit in den äußeren Bereichen des Mediums eine höhere Datenrate. Bereiche gleicher Datenmenge pro Spur werden zu Zonen zusammengefasst.

Literaturverzeichnis

[AAD97] ABOUTABL, Mohamed; AGRAWALA, Ashok; DECOTIGNIE, Jean-Dominique (1997): *Temporally Determinate Disk Access: An Experimental Approach.*

[BSG98] BARVE, Rakesh; SHRIVER, Elizabeth; GIBBONS, Phillip B.; HILLYER, Bruce K.; MATIAS, Yossi; VITTER, Jeffrey Scott (1998): *Modeling and optimizing I/O throughput of multiple disks on a bus.*

[BBD95] BECK, Michael; BÖHME, Harald; DZIADZKA, Mirko; KUNITZ, Ulrich; MAGNUS, Robert; VERWORNER, Dirk (1995): *Linux-Kernel-Programmierung*, 3. Auflage, Addison-Wesley

[BoCe01] BOVET, Daniel P.; CESATI, Marco (2001): *Understandig the Linux Kernel.* 1. Auflage, O'Reilly

[BrSch02] BREMER, Lars; SCHUSTER, Johannes (2002): *Platten-Karusell; Interne und externe Festplatten.* **In:** c't 8/02. Heise Verlag Hannover, Seite 194–203

[Boe96] BÖGEHOLZ, Harald (1996): *Platten-Karussell; Festplatten mit EIDE- und SCSI-Schnittstelle im Überblick.* **In:** c't April/96. Heise Verlag Hannover, Seite 268–274

[Boe97] BÖGEHOLZ, Harald (1997): *Platten-Karussell; Festplatten mit EIDE- und SCSI-Schnittstelle im Überblick.* **In:** c't 14/97. Heise Verlag Hannover, November 1997, Seite 160–169

[Coker] COKER, Russell: *Zoned Constant Angular Velocity testing program — ZCAV.* http://www.coker.com.au/bonnie++/zcav/ [Zugriff am 05.04.2002]

[GaSch00] GANGER, Greg; SCHINDLER, Jiri (2000): *Database of Validated Disk Parameters for DiskSim.* http://www.ece.cmu.edu/~ganger/disksim/diskspecs.html [Zugriff am 17.04.2002]

[GWP98] GANGER, Gregory R.; WORTHINGTON, Bruce L.; PATT, Yale N. (1998): *The DiskSim Simulation Environment, Version 1.0 Reference Manual.*

[Gil02] GILBERT, Douglas: *The Linux 2.4 SCSI subsystem HOWTO: raw devices.* http://www.linuxdoc.org/HOWTO/SCSI-2.4-HOWTO/rawdev.html [Zugriff 28.03.2002]

[HLR01] HAMANN, Cl.-J.; LÖSER, Jork; REUTHER, Lars; SCHÖNBERG, Sebastian; WOLTER, Jean; HÄRTIG, Hermann (2001): *Quality Assuring Scheduling - Deploying Stochastic Behavior to Improve Resource Utilization.* **In**: proceedings of the 22th

IEEE Real-Time Systems Symposium (RTSS-XXII), December 2001, London, UK

[IBM01] IBM (2001): *Hard disk drive specifications, Deskstar 60GXP, Revision 2.1. 27 August.*

[IyDr01] Iyer, Sitaram; Druschel, Peter (2001): *Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O.*

[Köh96] Köhntopp, Kristian (1996): *Dynamisierung im UNIX-Dateisystem.* Diplomarbeit, Institut für Informatik und Praktische Mathematik der Christian-Albrechts-Universität zu Kiel

[Max01] Maxtor (2001): *Product Support, Technical Notes : 21008.* http://www.maxtor.com/products/diamondmax/techsupport/technicalprocedures/21008.html [Zugriff am 18.04.2002]

[Meh98] Mehnert, Frank (1998): *Ein zusagenfähiges SCSI-Subsystem für DROPS.* Diplomarbeit, Institut für Betriebssysteme, Datenbanken und Rechnernetze, Fakultät Informatik, Technischen Universität Dresden

[Met97] Meter, Rodney Van (1997): *Observing the Effects of Multi-Zone Disks.* http://www.isi.edu/netstation/zcav/zcav.ps [Zugriff am 29.03.02]

[OCH85] Ousterhout, John K.; Costa, Hervé Da; Harrison, David; Kunze, John A.; Kupfer, Mike; Thompson, James G. (1985): *A Trace-Driven Analysis of the UNIX 4.2 BSD File System.* Computer Science Division, Electrical Engineering and Computer Sciences, University of California Berkeley

[Poh01] Pohlack, Martin (2001): *Kernmodul für Zeitmessung.* Komplexpraktikumsbericht, Institut für Systemarchitektur, Fakultät Informatik, TU-Dresden

[Reu98] Reuther, Lars (1998): *Entwicklung eines echtzeitfähigen Dateisystems.* Diplomarbeit, Technischen Universität Dresden, Fakultät Informatik, Institut für Betriebssysteme, Datenbanken und Rechnernetze Lehrstuhl für Betriebssysteme

[RueWi94] Ruemmler, Chris; Wilkes, John (1994): *An introduction to disk drive modeling.* Hewlett-Packard Laboratories, Palo Alto, CA

[Scha99] Schank, Andreas (1999): *Integration von Mechanismen zur Echtzeitunterstützung in das Linux-Filesystem.* Diplomarbeit, 6. Juni, Department of Computer Science, University of Kaiserslautern, Germany

[SchGa99] Schindler, Jiri; Ganger, Gregory R. (1999): *Automated Disk Drive Characterization.* Carnegie Mellon University

[SGL02] Schindler, Jiri; Griffin, John Linwood; Lumb, Christopher R.; Ganger, Gregory R. (2002): *Track-aligned Extents: Matching Access Patterns to Disk Drive Characteristics.*

[Sch94] SCHMIDT, Friedhelm (1994): *SCSI-BUS und IDE-Schnittstelle.* 1. Auflage 1993/1. korrigierter Nachdruck 1994, Addison-Wesley

[Shr97] SHRIVER, Elizabeth (1997): *Performance modeling for realistic storage devices.* dissertation

[SMW98] SHRIVER, Elizabeth; MERCHANTY, Arif; WILKES, John: *An analytic behavior model for disk drives with readahead caches and request reordering.*

[TAP00] TALAGALA, Nisha; ARPACI-DUSSEAU, Remzi H.; PATTERSON, David (2000): *Microbenchmark-based Extraction of Local and Global Disk Characteristics.* Technical report CSD 99 1063, University of California, Berkeley

[ATA01] Technical Committee T13 AT Attachment (Hrsg.) (2001): *Working T13 Draft 1410D Revision 3, Information Technology - AT Attachment with Packet Interface - 6 (ATA/ATAPI-6).* http://www.t13.org/project/d1410r3.pdf [Zugriff am 28.03.2002]

[Vog99] VOGELS, Werner (1999): *File system usage in Windows NT 4.0.* Cornell University

[WGPW96] WORTHINGTON, Bruce L.; GANGER, Gregory R.; PATT, Yale N.; WILKES, John (1996): *On-Line Extraction of SCSI Disk Drive Parameters.*