Diplomarbeit

Plattenscheduling für Quality-Assuring-Scheduling

Martin Pohlack

4. März 2003

Technische Universität Dresden
Fakultät Informatik
Institut für Systemarchitektur
Professur Betriebssysteme

Betreuender Hochschullehrer:	Prof. Dr. rer. nat. Hermann Härtig
Betreuender Mitarbeiter:	Dipl.-Inf. Lars Reuther

Danksagung

An erster Stelle gilt mein Dank Prof. Härtig für die Möglichkeit, in einer so guten Gruppe arbeiten zu können und die wertvollen Tipps und Anregungen während der Arbeit. Insbesondere möchte ich mich bei meinem Betreuer Lars Reuther für die vielfältige Unterstützung bedanken, die er mir gewährt hat.

Des Weiteren gilt mein Dank Frank Mehnert und Adam Lackorzynski für technische Unterstützung, aber auch dem Rest der Betriebssystemegruppe der TU Dresden.

Ein großes Dankeschön geht auch an meine Freundin Juliane Schröter, insbesondere für das erste Korrekturlesen dieser Arbeit und für viele wertvolle Hinweise.

Meinen Eltern danke ich besonders dafür, dass sie mir dieses Studium ermöglicht haben und für die nützlichen Hinweise zu diese Arbeit.

Zum Abschluss möchte ich mich noch ganz herzlich bei all jenen bedanken, die weiterhin zum Erfolg dieser Arbeit beigetragen haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Das *Dresden Real-Time Operating System* (DROPS) (vgl. [DT02]) ist ein Forschungsprojekt der Betriebssystemegruppe der Technischen Universität Dresden. Es basiert auf dem Mikrokern L4/Fiasco. Ein Hauptziel von DROPS ist die Unterstützung von Anwendungen mit *Quality-of-Service*-Anforderungen.

Grundlage dafür ist die Echtzeitfähigkeit des Betriebssystemkerns und der verwendeten Komponenten. *Quality Assuring Scheduling* (QAS) ist dabei eine wesentlicher Bestandteil. Durch eine geringe Reduktion der Qualitätsanforderungen kann oftmals eine starke Steigerung der zusicherbaren Bandbreite erreicht werden (vgl. [HLR+01]). So ist zum Beispiel bei einem Videostrom eine Qualität von 95% oftmals vollkommen ausreichend. Das menschliche Sehvermögen reagiert sehr tolerant, wenn einige Einzelbilder fehlen. Demgegenüber ist die starke Erhöhung der durchschnittlichen Bandbreite des Videostromes, die durch die leichte Qualitätsreduktion ermöglicht wird, deutlich gütesteigernd zu bemerken.

Auf Unterbrechungen in Audioströmen reagiert das menschliche Hörvermögen dagegen sehr viel empfindlicher. Deshalb ist hier eine höhere Qualitätsstufe, bis hin zu 100%, angebracht.

Im Rahmen von DROPS wird auch an einem echtzeitfähigen Dateisystem gearbeitet, welches gleichzeitig Datenströme mit Echtzeitanforderungen und herkömmlichen Dateizugriff unterstützt und damit als Datenquelle im Sinne von QAS fungieren soll. Dieses Dateisystem stützt sich auf einen echtzeitfähigen Plattenauftragsscheduler.

Ziel dieser Diplomarbeit ist es, den bisherigen Scheduler durch eine verbesserte Version zu ersetzen. Dieser soll unter Einhaltung der Echtzeitschranken die Bearbeitungsreihenfolge der Festplattenaufträge mit dem Ziel der Leistungssteigerung verändern. Dabei sind die widersprüchlichen Ziele *möglichst hohe Plattenleistung* und *Einhaltung der Echtzeitzusagen* in Einklang zu bringen.

Besonders beim Scheduling von Festplattenaufträgen hat die Reihenfolge, in der die einzelnen Aufträge abgearbeitet werden, extreme Auswirkungen auf die Bearbeitungsdauer. Die Zugriffszeit hängt stark von der Position der Daten auf der Festplatte ab. Die Untersuchung von Plattenauftrags-Schedulingalgorithmen ist deshalb seit vielen Jahren Forschungsgegenstand (zum Beispiel [One75, Wil76]).

Die vorliegende Arbeit baut auf einer Analyse bekannter Schedulingalgorithmen auf. Basierend auf dieser Analyse wird ein möglichst leistungsstarkes Verfahren zur Implementierung ausgewählt.

Die Beschreibung der Implementierung selbst und der Anpassung an QAS bilden einen weiteren Schwerpunkt dieser Arbeit. Zusammenfassend folgt eine Bewertung des implementierten Verfahrens.

Gliederung

Das folgende Kapitel enthält eine Einführung in das Thema. Grundlagen werden erläutert und auf verwandte Arbeiten wird hingewiesen. Kapitel 3 geht auf Entwurfsziele und Schedulingverfahren ein. Im darauf folgenden Kapitel werden Besonderheiten der Implementierung beschrieben. Bewertet wird diese in Kapitel 5. Außerdem werden dort die Messergebnisse kommentiert. Mit Zusammenfassung und Ausblick schließt die Arbeit.

2 Stand der Technik

Das folgende Kapitel ist in vier Bereiche unterteilt. Zuerst soll, beginnend bei ihrem physischen Aufbau, eine grundlegende Einführung in die Thematik der Festplatten gegeben werden. Danach folgt eine kurze Beschreibung des zeitlichen Ablaufes eines typischen Plattenauftrages. Der erste Teil schließt mit einer Diskussion von *Tagged Command Queueing*.

Der zweite Teil führt kurz in einige grundlegende Begriffe von Echtzeitsystemen ein. Außerdem wird die Idee des *Quality Assuring Schedulings* (QAS) [HLR^{+}01] erläutert.

Im dritten Teil des Kapitels wird der SCSI-Plattentreiber von DROPS beschrieben. Er bildet die Basis für die vorliegende Arbeit. Im Besonderen wird dabei auf die Architektur innerhalb von DROPS eingegangen.

Das Kapitel schließt im vierten Teil mit einem knappen Überblick thematisch verwandter Arbeiten.

2.1 Festplatten

2.1.1 Aufbau

Der physische Aufbau von Festplatten ist ein wichtiger Schlüssel zum Verständnis ihres Verhaltens, da selbiges von mechanischen Vorgängen dominiert wird. Eine Einführung in diesen Bereich scheint deshalb notwendig.

Festplatten bestehen aus einem Stapel von magnetisch beschichteten Platten, die auf einer gemeinsamen Achse gelagert sind. Beide Oberflächen der Platten können zur Datenspeicherung benutzt werden. Jede Oberfläche des Plattenstapels wird von einem eigenen Schreib-Lese-Kopf bearbeitet. Diese Köpfe sind an einem gemeinsamen Kamm befestigt und werden von einem Linearmotor zwischen äußerem und innerem Rand der beschichteten Oberflächen bewegt.

Der Plattenstapel rotiert im normalen Betrieb mit einer konstanten Winkelgeschwindigkeit. Festplatten im Desktop-Bereich erreichen heute 5.400 beziehungsweise 7.200 u/min. Für Server setzt man Platten mit bis zu 15.000 u/min ein.

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, von denen jeder

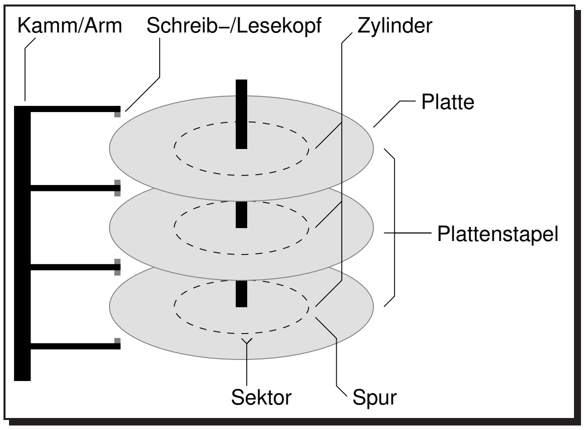


Abbildung 2.1: Aufbau einer Festplatte

gleiche Mengen an Nutzdaten aufnehmen kann. Bei heutigen Festplatten sind das typischerweise 512 Byte. Bei älteren Modellen waren auch 256 und 1024 Byte üblich [RW94, Meh98].

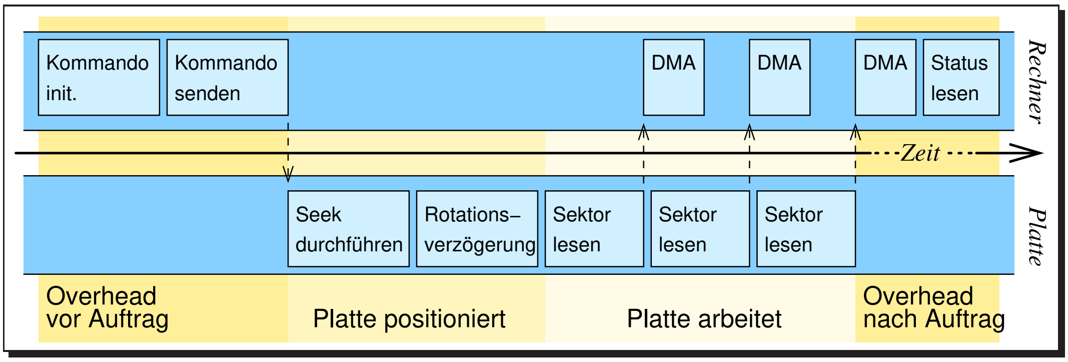
Die Spuren, die auf den einzelnen Oberflächen über den gleichen Radius verfügen, bilden zusammen einen Zylinder. Gruppen von Zylindern, deren Spuren genauso viele Sektoren enthalten, werden zu Zonen zusammengefasst. Typischerweise können Festplatten bei sequentiellem Zugriff in verschiedenen Zonen unterschiedliche Datenraten liefern, da die Sektorzahl pro Rotationswinkel unterschiedlich ist. Die schnellen Zonen liegen normalerweise am äußeren Rand der Platte, da dort am meisten Sektoren pro Spur untergebracht werden können.

Ein Sektor ist die kleinste auf Festplatten adressierbare Einheit. Heute kommt vorwiegend das LBA-Verfahren (*logical block addressing*) zum Einsatz, bei welchem die einzelnen Sektoren über eine logische Sektoradresse angesprochen werden. Die Abbildung der logischen Adressen auf die physischen Positionen geschieht in der Platte und wird oftmals als Mapping bezeichnet. Dabei können verschiedene Verfahren zum Einsatz kommen, welche in [Poh02] näher beschrieben sind. Im Rahmen der vorliegenden Arbeit ist vor allem das lineare Mapping (auch *vertikales* Mapping) interessant. Dabei werden niedrige logischer Adressen auf den äußeren Plattenrand abgebildet. Mit steigenden logischen Adressen werden weiter innen liegende Bereiche der Platte adressiert. Die Sektoren der Spuren eines Zylinders haben also nahe beieinander liegende logische Adressen.

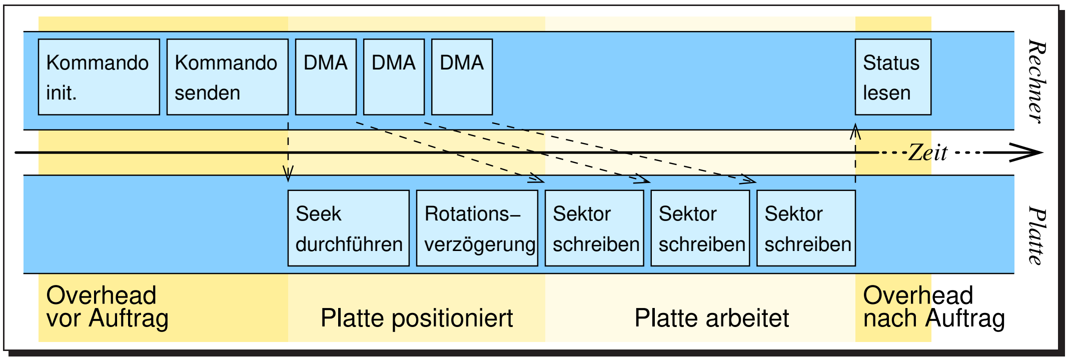
2.1.2 Zeitlicher Ablauf eines Plattenauftrages

In Abbildung 2.2 ist die Aufteilung eines Plattenauftrages in vier Zeitabschnitte dargestellt:

1. Zu Beginn des Auftrages entsteht eine Verzögerung, die für die Kommunikation des Kontrollers mit der Platte notwendig ist. Das Kommando wird in diesem Abschnitt initialisiert und an die Platte geschickt.



(a) Leseauftrag



(b) Schreibauftrag

Abbildung 2.2: Zeitablauf von Plattenaufträgen mit drei Zielsektoren

2. Wenn die Platte das Kommando empfangen hat, wird der Plattenarm in das Zielgebiet bewegt:

   - Dazu muss der Arm im Regelfall eine Seek-Bewegung durchführen.
   - Danach schließt sich eine Periode an, in der die Platte auf den Zielsektor wartet — die Rotationsverzögerung.

   Diese zweite Phase ist der Optimierungsgegenstand von klassischen Schedulingalgorithmen. Mit den meisten Algorithmen wird versucht, den ersten Anteil dieser Phase (Seek-Bewegung) zu minimieren.

3. Die dritte Phase ist die eigentliche Arbeitsphase der Platte. Daten werden vom Datenträger gelesen beziehungsweise auf ihn geschrieben, während die Zielsektoren unter dem Schreib-Lese-Kopf entlangrotieren. Je länger der Anteil dieser Phase an der Gesamtausführungszeit des Plattenauftrages ausfällt, desto höher ist die Auslastung der Platte.

4. Der letzte Zeitanteil eines Plattenkommandos ist der Abschluss. Hier wird je nach Auf-

tragstyp (Lese-/Schreibauftrag), entweder der letzte Rest der Daten transferiert oder nur eine Fertigmeldung an den Kontroller gegeben.

Der dargestellte Ablauf gilt für Aufträge, die nicht aus dem Platten-Cache befriedigt werden können. Wenn die Festplatte *"Access-On-Arrival"* unterstützt, entfällt unter Umständen ein Teil der Rotationsverzögerung oder tritt später auf.

2.1.3 Tagged Command Queueing

Ohne *Tagged Command Queueing (TCQ)* schickt der Plattentreiber einen neuen Plattenauftrag erst an die Platte, nachdem der vorherige abgeschlossen wurde[1]. Diese Vorgehensweise lässt sich zwar einfach implementieren, hat aber den Nachteil, dass die Platte in der Übertragungszeit des neuen Kommandos[2] und in der Abschlusszeit des vorhergehenden[3] nicht ausgelastet ist und damit leer läuft.

TCQ ermöglicht es nun bereits vor Abschluss eines Kommandos weitere an die Platte zu schicken. Die Platte hat sogar die Möglichkeit, die Abarbeitungsreihenfolge der schon an sie geschickten Kommandos zu verändern, das heißt sie kann selbst planen.

Bei TCQ unterscheidet man drei verschiedene Auftragstypen, die die Abarbeitungsreihenfolge in der Platte beeinflussen (vgl. [Tec96]):

- HEAD OF QUEUE TAG: Aufträge dieses Typs werden bei Eintreffen in der Platte an die Spitze der Warteschlange gestellt und als nächstes abgearbeitet. Wenn mehr als ein Auftrag dieses Typs vorliegt werden sie gemäß LIFO (*last in first out*) abgearbeitet.

- ORDERED QUEUE TAG: Diese Aufträge werden ganz normal hinten in die Warteschlange eingereiht. Andere Aufträge (ORDERED oder SIMPLE) können diese aber bei Umsortierung nicht überholen und nicht von diesen überholt werden. Sie stellen damit eine Barriere in der Warteschlange dar. Werden nur Aufträge dieses Typs benutzt, wird das interne Scheduling der Platte praktisch deaktiviert, da alle Aufträge nach FIFO (*first in first out*) bearbeitet werden müssen.

- SIMPLE QUEUE TAG: Bei Aufträgen dieses Typs gibt es, bis auf die für die beiden vorher genannten Typen geltenden, keine Einschränkungen bezüglich der Abarbeitungsreihenfolge. Werden nur derartige Aufträge benutzt, kann der interne Plattenscheduler frei und damit zum Beispiel optimal bezüglich Auslastung entscheiden.

TCQ bietet also im Wesentlichen zwei Vorteile. Der erste ist die Ausnutzung der Übertragungszeit zwischen Platte und Kontroller, um andere Aufträge auszuführen. Der zweite Vorteil ist die

[1] das heißt nach Phase 4 eines Kommandos folgt immer Phase 1 des nächsten (siehe Abschnitt 2.1.2)
[2] also Phase 1 (siehe Abschnitt 2.1.2)
[3] also Phase 4 (siehe Abschnitt 2.1.2)

Möglichkeit, dass die Platte selbst schedulen und damit ihre Auslastung erhöhen kann. Das ist sinnvoll, da die Platte die genauesten Informationen über ihren Zustand hat und damit theoretisch die beste Schedulingentscheidung treffen kann.

Dieser Vorteil ist aber gleichzeitig auch ein Nachteil, da durch das interne Scheduling die Abschätzung von Worst-Case-Bearbeitungszeiten verhindert wird. Damit ist ein Einsatz in Echtzeitsystemen *nicht* möglich.

Um gleichzeitig die Vorteile des minimierten Kommando-Overheads zu nutzen und nicht die Nachteile der verhinderten Echtzeitgarantien in Kauf nehmen zu müssen, bieten sich zwei Möglichkeiten an:

1. Die Warteschlangengröße in der Platte wird auf zwei beschränkt, sodass immer ein Auftrag bearbeitet wird und die Platte niemals zwei bereite Aufträge in der Warteschlange hat. Dadurch kann sie praktisch nicht umsortieren.

2. Alle Aufträge werden mit ORDERED QUEUE TAGs an die Platte geschickt. Dadurch wird ihr eine Umsortierung verboten.

2.2 Echtzeitsysteme

Für das weitere Verständnis dieser Arbeit sollen hier ein paar Grundbegriffe aus dem Bereich der Echtzeitsysteme eingeführt werden. Diese und weitere Begriffe können zum Beispiel in [Liu00] nachgeschlagen werden.

2.2.1 Scheduling

Eine Aufgabe moderner Betriebssysteme ist die Verwaltung der verschiedenen Ressourcen, zum Beispiel Speicher, Prozessorzeit und Plattenbandbreite. Üblicherweise konkurrieren einzelne Prozesse um diese Ressourcen. Ein bestimmter Teil des Betriebssystems — der Scheduler — ist für die Zuteilung der Ressourcen zuständig. Normalerweise werden für unterschiedliche Ressourcen unterschiedliche Scheduler benutzt. Ein Prozessorscheduler bestimmt zum Beispiel, welcher Prozess als nächster auf dem Prozessor rechnen darf, ein Plattenauftragsscheduler berechnet die Reihenfolge, in der die einzelnen Aufträge an die Platte weitergereicht werden.

Der wesentliche Unterschied zwischen Plattenauftrags- und Prozessorzeitscheduling ist die Nicht-Unterbrechbarkeit von Plattenaufträgen. Sind sie einmal an die Platte abgeschickt, so werden sie fertig abgearbeitet und erst danach können andere Aufträge erfüllt werden. Ein rechnender Prozess hingegen kann von einem modernen Betriebssystem an (fast[4]) jeder Stelle unterbrochen und später fortgesetzt werden.

[4] Eine Ausnahme ist zum Beispiel der X-Server `xfree86` in Linux, der selbst Interrupts sperren und seine Unterbrechbarkeit damit aufheben kann.

Für das Scheduling existiert ein Reihe von unterschiedlichen Algorithmen, mit denen verschiedene Ziele erreicht werden können. Aus Sicht des Systembetreibers ist zum Beispiel der maximale Durchsatz interessant, wohingegen der einzelne Nutzer an möglichst schnellen Antwortzeiten interessiert ist.

Im Kontext von Echtzeitsystemen kommt ein weiteres Ziel hinzu — die Einhaltung von Zeitschranken für einzelne Aufträge.

2.2.2 Admission

Damit ein Scheduler in einem Echtzeitsystem überhaupt die Möglichkeit hat, alle Zeitschranken einzuhalten, ist eine Reservierung der benötigten Ressourcen notwendig. Das muss vor der Nutzung und mit einer entsprechenden Beschreibung der gewünschten Ressource und Qualität geschehen.

Reservierungen müssen nicht in jedem Fall erfolgreich verlaufen, schließlich kann das System bereits ausgelastet sein. Über die Zulassung einer neuen Reservierung entscheidet die Admission (auch *acceptance test*). Im einfachsten Fall wird dabei nur eine Ja/Nein-Entscheidung getroffen. Im Rahmen des QAS gehört zur Admission außerdem die Berechnung der Zeitkonten für die Nutzung der angeforderten Ressource (vgl. [HLR+01]). Das Scheduling erfolgt bei QAS periodisch, das heißt das berechnete Zeitkonto gilt pro Periode.

Für die Reservierung eines Echtzeitstromes von der Festplatte sind beispielsweise die Anzahl der Plattenaufträge pro Periode, die Größe der Aufträge und die gewünschte Qualitätsstufe (also der Anteil der zu erfüllenden Aufträge) anzugeben.

Zusammen mit der Dichteverteilung der Bearbeitungszeiten von Plattenaufträgen kann von der Admission ein Zeitkonto berechnet werden. Dieses Zeitkonto wird bei QAS während des Schedulings benutzt. Zur Laufzeit wird das Konto jedes Stromes um die Zeitbeträge vermindert, die zur Bearbeitung der zugehörigen Aufträge benötigt wurden. Wenn das Zeitkonto in einer Periode aufgebraucht ist, wird vom Scheduler kein zum entsprechenden Strom gehörender Auftrag mehr ausgewählt.

Wie weiter oben schon erwähnt wurde, werden die Konten zum Periodenbeginn zurückgesetzt.

2.2.3 Worst-Case-Ausführungszeit

Bei Echtzeitsystemen spielt der Begriff der Worst-Case-Ausführungszeit eine wichtige Rolle. Er steht für die denkbar längste Ausführungszeit eines bestimmten Auftrages. Der Scheduler muss oft mit der Worst-Case-Ausführungszeit planen.

Bei Festplattenaufträgen hat die Worst-Case-Ausführungszeit eine besondere Bedeutung, da begonnene Aufträge nicht abgebrochen werden können. Aufträge, die zu lange dauern, verpassen somit nicht nur ihre eigene Zeitschranke, sondern können auch nachfolgende Aufträge verzögern. In [Poh02] wird eine Methode zur Ermittlung der Worst-Case-Ausführungszeiten von Festplattenaufträgen beschrieben.

2.3 Aktuelle Situation im DROPS-SCSI-Treiber

Die erste Version des SCSI-Subsystems für DROPS wurde von Frank Mehnert implementiert und in [Meh98] genauer beschrieben. Die aktuelle Version stammt von Lars Reuther. Sie ist eine Portierung aus dem Linux-Kern und läuft in DROPS als normaler Nutzerprozess.

2.3.1 Auftragsklassen

Das Subsystem unterstützt drei Klassen von Plattenaufträgen, wobei Aufträge mit Echtzeitanforderungen einzelnen Strömen zugeordnet sind:

1. Die erste Klasse bilden Plattenaufträge von Echtzeitströmen, die zu 100% zu erfüllen sind. Aufträge dieser Klasse müssen mit ihren Worst-Case-Bearbeitungszeiten reserviert und eingeplant werden. (vgl. *mandatory parts* in [HLR$^+$01])

2. Von den Aufträgen, die zu Strömen der zweiten Klasse gehören, ist nur ein bestimmter Anteil zu erfüllen. Für diese Ströme kann eine viel höhere Bandbreite zugesichert werden, da ihre Aufträge nur mit durchschnittlicher Bearbeitungszeit eingeplant werden müssen. Typische Vertreter dieser Klasse sind zum Beispiel MPEG-Videoströme. (vgl. *optional parts* in [HLR$^+$01])

3. Nicht-Echtzeitaufträge haben keine Zeitschranken und werden folglich mit niedrigster Priorität eingeplant. Dieses Vorgehen hat eine schlechte mittlere Antwortzeit für diese Aufträge zur Folge.

Aufträge der ersten beiden Klassen werden Zeitperioden zugeordnet, in denen sie abgearbeitet werden müssen. Zeitschranke ist für sie immer das Periodenende.

Alle Aufträge einer Klasse werden nach dem FIFO-Prinzip eingeplant. Die Bearbeitung beginnt mit den Aufträgen der Klasse 1. Danach werden alle Aufträge der Klasse 2 und zum Schluss einer Periode alle Nicht-Echtzeitaufträge, die noch in die Periodenrestzeit passen, abgearbeitet. Nicht bearbeitete Echtzeitaufträge verfallen am Periodenende, Nicht-Echtzeitaufträge bleiben unabhängig davon erhalten.

Diese Art der Bearbeitung garantiert die zugesagten Zeitschranken, verschenkt aber viel Plattenbandbreite, da bei keiner Schedulingentscheidung die Plattenauslastung berücksichtigt wird.

2.3.2 Architektur

Abbildung 2.3 illustriert die Einordnung des SCSI-Plattentreibers in DROPS. Sowohl auf der Ebene des Plattentreibers als auch auf der Ebene der Hardware gibt es Auftragswarteschlangen. Nutzerprozesse[5] wenden sich mit ihren Anfragen an den generischen Blocktreiber. Dieser reicht die Anfragen an konkrete Gerätetreiber weiter, zum Beispiel den SCSI-Plattentreiber.

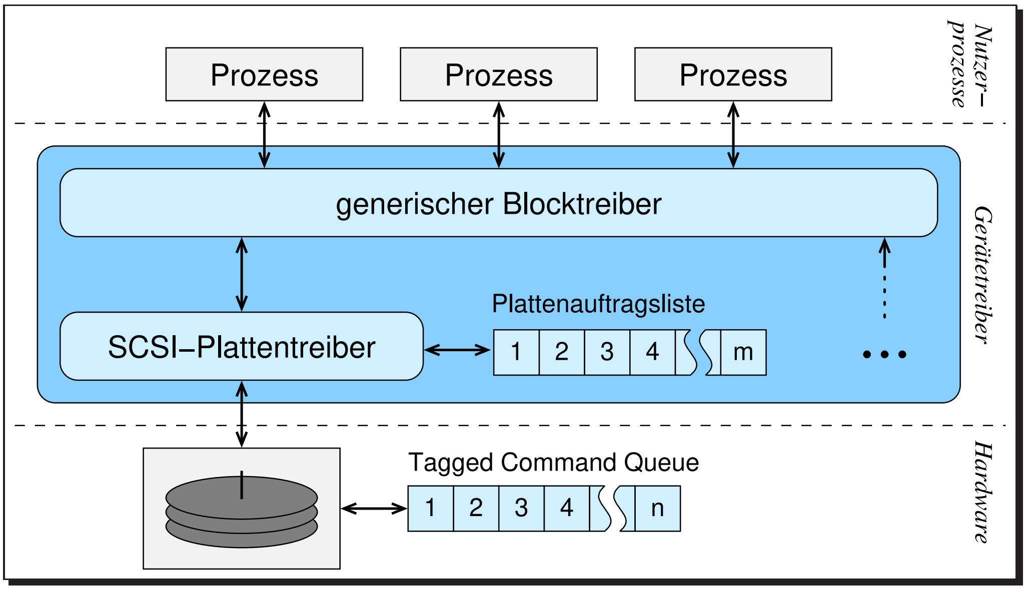


Abbildung 2.3: Schematischer Aufbau aus Treiber und Platte in DROPS

2.4 Verwandte Arbeiten

2.4.1 Disk scheduling revisited

SELTZER ET AL. untersuchen in [SCO90] verschiedene Schedulingverfahren bezüglich Plattenauslastung und Fairnesskriterien. In dieser Arbeit wird Algorithmen, die versuchen, die Rotationsverzögerung bei Plattenzugriffen zu minimieren, ein großes Potential zugesprochen. Außerdem werden verschiedene Möglichkeiten zur Modifikation der gegebenen Algorithmen diskutiert, wobei faire Behandlung der einzelnen Aufträge im Mittelpunkt steht:

- **Gruppierung**: Plattenaufträge werden gemäß ihrer Zielposition auf der Platte zu Gruppen zusammengefasst. Innerhalb dieser Gruppen wird der Schedulingalgorithmus normal angewandt. Die einzelnen Gruppen werden nacheinander abgearbeitet. Über die Gruppengröße und Aufteilung lässt sich die Auslastung der Platte und die maximale Bearbeitungszeit der einzelnen Aufträge steuern.

[5] Das kann zum Beispiel ein Dateisystemserver sein, dessen Dateisystem auf einer SCSI-Platte liegt.

- **Altern** (*aging*): Zur Auswahl eines zu schedulenden Plattenauftrages wird neben dem durch den konkreten Algorithmus vorgegebenen, herkömmlichen Kriterium ein Zweites herangezogen. Dieses zweite Kriterium ist die Verweildauer des Plattenauftrages in der Warteschlange. Beide werden gewichtet. Die Wahrscheinlichkeit, dass ein Auftrag ausgeführt wird steigt also mit seiner Verweildauer in der Warteschlange an. Bei diesem Verfahren lässt sich Auslastung und maximale Bearbeitungszeit mithilfe einer Wichtungsfunktion steuern.

Die Messwerte der komplizierteren Algorithmen in der Veröffentlichung resultieren aus Simulationen.

2.4.2 Disk scheduling algorithms based on rotational position

JACOBSON und WILKES untersuchen in [JW91] verschiedene Schedulingalgorithmen, die die Rotationsverzögerung minimieren sollen. Eine wichtige Rolle spielt dabei die Verhinderung von Verhungern (*starvation*), also die Steuerung der maximalen Bearbeitungszeit für einzelne Aufträge. Gegenüber der im vorigen Abschnitt besprochenen Arbeit kommt hier die Idee des *"Batch-Algorithmus"* hinzu. Das Anhängen neuer Aufträge an die Auftragswarteschlange wird bis zu ihrer vollständigen Abarbeitung unterbrochen. Damit wird ein beliebiges Verzögern einzelner Aufträge verhindert. Nachteil ist die Reduktion der durchschnittlichen Warteschlangengröße, die der Scheduler zur Verfügung hat.

Die vorgestellten Ergebnisse zeigen, dass mit rotationssensitiven Schedulingalgorithmen prinzipiell eine höhere Plattenauslastung erreicht werden kann. Sie basieren aber komplett auf der Simulation einer Festplatte.

2.4.3 Temporally Determinate Disk Access: An Experimental Approach

ABOUTABLE ET AL. stellen in [AAD98] ein Plattenmodell vor, das es ermöglichen soll, die Bearbeitungszeit von Plattenaufträgen vorherzuberechnen. Bestandteil dieses Modells sind unter anderem die Seek-Funktion, der Spurversatz, der Kontroller-Overhead und Kopf- sowie Spurumschaltzeiten. Verschiedene Verfahren zur Ermittlung der einzelnen Plattenmodellparameter werden erwähnt.

Ansätze des in dieser Arbeit entwickelten Plattenmodells werden auch in der vorliegenden Arbeit benutzt. Das betrifft zum Beispiel die Modellierung der Bearbeitungszeit von Plattenaufträgen als eine Summe von Einzelzeiten. ABOUTABLE ET AL. identifizieren in ihrer Arbeit vier Summanden: Die Seek-Verzögerung (*seek delay*), die Rotationsverzögerung (*rotational delay*), die Medienzugriffszeit (*off-surface transfer delay*) und die Übertragungsverzögerung (*host transfer delay*). Im Rahmen der vorliegenden Arbeit wird der vierte Summand als Overhead-Anteil bezeichnet. Außerdem hat es sich als vorteilhaft erwiesen, ihn in weitere Einzelkomponenten zu zerlegen.

ABOUTABLE ET AL. verifizieren die erreichte Modellgenauigkeit anhand von Simulationen.

2.4.4 Andere

In [HC02] kommen HUANG und CHIUEH zu dem Schluss, dass rotationssensitive Schedulingalgorithmen nicht praktikabel sind, da sich einmal entwickelte Plattenmodelle nicht gut auf andere Platten übertragen lassen. Sie gehen davon aus, dass sich Plattenauftragsbearbeitungszeiten nicht genau genug vorhersagen lassen. Außerdem wird argumentiert, dass von derartigen Schedulingalgorithmen keine wesentliche Leistungssteigerung zu erwarten ist.

In der vorliegenden Arbeit wird ein *anderer* Schluss gezogen, sowohl bezüglich der Realisierbarkeit als auch des Leistungsgewinns.

WORTHINGTON ET AL. untersuchen in [WGP94] die Auswirkung von komplexeren Eigenschaften moderner Festplatten (zum Beispiel Zonen und Caches) auf bekannte Schedulingalgorithmen. Algorithmen werden unter verschiedenen synthetischen und realen Systemlasten bewertet. Außerdem wird der Leistungsgewinn, der durch vollständige Geometrieinformationen erreicht werden kann, untersucht. Bei Seek-Zeit-minimierenden Algorithmen scheint der Gewinn vernachlässigbar, wohingegen positionierungszeitminimierende Algorithmen erst damit möglich sind, und gleichzeitig die größte Leistung versprechen.

3 Entwurf

Das Entwurfskapitel gliedert sich in drei Teilbereiche. Zuerst werden noch einmal die beiden Hauptziele — Auslastung und Echtzeit — erläutert. Der zweite Teil stellt einige Plattenauftrags-Schedulingalgorithmen vor und geht kurz auf Voraussetzungen und zu erwartende Leistung ein. Außerdem wird ein Algorithmus ausgewählt und verschiedene Voruntersuchungen für dessen Implementierung werden dargestellt. Im dritten Teil wird ein Verfahren skizziert, das es ermöglicht Echtzeitzusagen einzuhalten und gleichzeitig eine möglichst hohe Plattenauslastung zu erreichen.

Die Plattenleistung und die Plattenauslastung sind eng aneinander gekoppelte Begriffe, da eine hohe Auslastung praktisch eine hohe Leistung bedeutet. Sie werden deshalb im Folgenden gleichberechtigt benutzt.

3.1 Entwurfsziele

Die Leistungssteigerung beim Festplattenzugriff, die durch geeignete Schedulingalgorithmen zu erreichen ist, fällt um so größer aus, je größer die Menge der bereiten Aufträge ist, aus denen ausgewählt werden kann.

Auf der anderen Seite werden von Echtzeitsystemen, wovon DROPS ein typischer Vertreter ist, gewisse Bearbeitungszeitgarantien für einzelne Aufträge gefordert. Es ist also nicht immer möglich, den für die Gesamtauslastung der Platte günstigsten Auftrag zur Bearbeitung auszuwählen. Hier ist eine Optimierungsaufgabe mit Randbedingungen zu lösen. Scheduler, die dieses nicht berücksichtigen, können durch bestimmte Schedulingentscheidungen gegebene Zeitgarantien verletzen.

Ein wichtiges Entwurfsziel war es deshalb, eine Kombination aus einem sehr guten Schedulingalgorithmus und einem Verfahren, welches gegebene Zeitgarantien einhält, zu finden.

3.1.1 Auslastung

Um eine Festplatte optimal auszulasten, gilt es ihre Leerlaufzeiten zu minimieren, also die Zeitanteile mit echtem Medienzugriff zu maximieren. Die größte Auslastung[1] ist dann zu erwarten,

[1] abgesehen von Zugriffen auf im Platten-Cache befindliche Daten

wenn sequentiell auf die Platte zugegriffen wird, da dabei keine unnötigen Kopfbewegungen notwendig sind.

Ein weiterer Parameter, der bei dieser Zugriffsart minimiert wird, ist die Rotationsverzögerung, da alle Zielsektoren direkt hintereinander auf der Platte liegen. Wenn der Schreib-Cache oder TCQ verwendet wird, kann sogar der Kommando-Overhead teilweise oder vollständig eliminiert werden. In den Übertragungszeiten werden andere Aufträge abgearbeitet. In dieser Situation kann die Platte also ihre theoretisch maximale Datenrate liefern. Die Datenrate, die eine Festplatte bei sequentiellem Zugriff liefert, ist somit ein oberer Grenzwert, der von Schedulingalgorithmen angestrebt werden sollte.

Zeitanteile eines Plattenauftrages, die Schedulingalgorithmen minimieren können, sind also:

- Kommando-Overhead (unterteilt in zwei Anteile):
  - Auftragsanfangs-Overhead und
  - Auftragsende-Overhead,
- Kopf-Positionierzeit (bestehend aus zwei Anteilen):
  - Seek-Zeit und
  - Rotationsverzögerung.

Weitere Optimierungsmöglichkeiten für die Plattenleistung, wie zum Beispiel die Benutzung von Caches im Betriebssystem und das Zusammenfassen vieler kleiner benachbarter Aufträge zu wenigen großen, sollen hier nicht weiter betrachtet werden, da diese Verfahren unabhängig vom verwendeten Scheduler zusätzlich eingesetzt werden können.

3.1.2 Einhaltung der Echtzeitgarantien

Maximale Plattenauslastung und Echtzeitgarantien für bestimmte Aufträge sind zwei Ziele, die sich nicht immer gleichzeitig erreichen lassen. Für maximale Auslastung muss der Scheduler die Bearbeitungsreihenfolge der Aufträge uneingeschränkt umsortieren können. Dadurch können jedoch einzelne Aufträge theoretisch beliebig verzögert werden und somit ihre Zeitschranken verpassen.

Eine Möglichkeit, gute Auslastung unter Beachtung der Echtzeitgarantien zu erreichen, besteht darin, den Scheduler nicht mehr beliebig aus der Auftragswarteschlange auswählen zu lassen, sondern eine Vorauswahl zu treffen. Das genaue Verfahren ist in Abschnitt 3.3 dargestellt.

3.2 Algorithmen

Wie in Abschnitt 2.1.2 erläutert, verbringt die Festplatte normalerweise wenig Zeit damit, Daten vom und zum Datenträger zu transferieren. Viel Zeit wird mit Warten auf Datenübertragung, Armbewegungen beziehungsweise die Rotationsverzögerung verbracht. Den Zeitanteil, den die Platte effektiv mit dem Medienzugriff verbringt, kann man vergrößern, indem man die Aufträge in einer bestimmten Reihenfolge und zu bestimmten Zeitpunkten an die Platte weiterreicht. Für die Bestimmung dieser Reihenfolge ist der Schedulingalgorithmus zuständig. Er wählt zu Schedulingzeitpunkten aus der Menge der verfügbaren (bereiten) Aufträge einen Kandidaten aus. Die Menge dieser Aufträge ist üblicherweise in einer Liste oder einer ähnlichen Datenstruktur zusammengefasst und wird im Weiteren als Warteschlange bezeichnet. Die Größe der Warteschlange kann einen großen Einfluss auf die Auslastung der Platte haben.

Im Folgenden werden einige grundlegende, aber auch komplexere Algorithmen eingeführt:

- **FIFO** steht für *first in first out* und bedeutet, dass Aufträge in der Reihenfolge ausgeführt werden, in der sie in die Warteschlange eingereiht wurden. Dieser Algorithmus ist extrem einfach zu realisieren, da *immer* das erste Element aus der Warteschlange an die Platte weitergereicht wird.

  Andererseits ist kein Gewinn bei der Plattenauslastung zu erwarten, da keinerlei Veränderung der Reihenfolge vorgenommen wird. Für diesen Algorithmus ist auch kein Wissen über die Platte und ihren aktuellen Zustand notwendig.

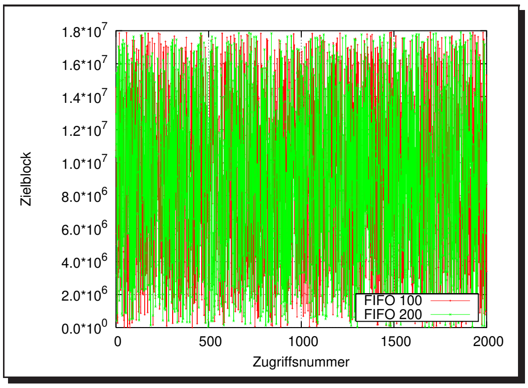


Abbildung 3.1: Dargestellt sind die Zielblöcke der Plattenaufträge, in der Reihenfolge wie sie vom Algorithmus an die Platte weitergereicht werden. FIFO sortiert keine Aufträge um, deshalb entspricht das Muster dem zufälligen Eingangsmuster. Eine Vergrößerung der Warteschlange von 100 auf 200 ändert die Art des Zufallsmusters nicht.

- Bei **SCAN** (auch *elevator*) wird versucht die Armbewegung der Platte zu minimieren. Der Arm führt bei diesem Schedulingverfahren eine Art Zick-Zack-Bewegung aus, indem er erst bis zum Plattenninneren bewegt wird und danach wieder bis zum äußeren Rand fährt.

Das wird erreicht, indem die einzelnen Aufträge so aus der Warteschlange genommen werden, dass ihre Zielzylinderadressen aufsteigend (bei Inwärtsbewegungen) beziehungsweise absteigend (bei Auswärtsbewegungen) sortiert sind. Wenn die Platte mit linearem Mapping[2] betrieben wird, erreicht man fast den selben Effekt mit einer Sortierung nach der logischen Blockadresse der Aufträge (vgl. [WGP94]). Diese ist — im Gegensatz zur physischen Adresse — normalerweise im Plattentreiber verfügbar.

Ein Gewinn bei der Plattenauslastung ist besonders dann zu erwarten, wenn der Zeitanteil, den die Platte mit Armbewegungen verbringt, sehr groß ist, da dieser durch den Algorithmus minimiert wird.

- Bei **cSCAN** (*circular* SCAN) wird der Plattenarm im Unterschied zu SCAN immer nur in eine Richtung bewegt. Am Ende der Bewegung erfolgt ein langer Rücksprung.

  Für die Plattenauslastung ist der Unterschied zwischen SCAN und cSCAN normalerweise nicht relevant. Bei Verwendung von SCAN ist für Aufträge am äußeren beziehungsweise inneren Plattenrand aber ein schlechteres Antwortverhalten zu erwarten als für Aufträge in den mittleren Bereichen, da die Ränder nicht in so gleichmäßigen Zeitabständen überstrichen werden wie das Zentrum. Bei cSCAN tritt dieses Problem nicht auf. Dafür muss ein langer Rücksprung in Kauf genommen werden. [SCO90]

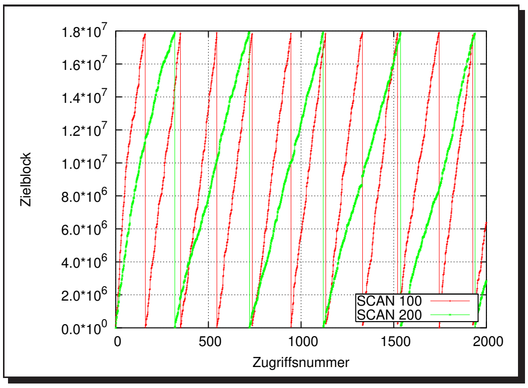


Abbildung 3.2: Bei cSCAN sind die aufsteigenden Zielblockadressen und die Rücksprünge sehr schön zu erkennen. Wird die Warteschlangengröße auf 200 gesteigert, vergrößern sich auch die Intervalle.

- Der **SSTF**-Algorithmus (*shortest seek time first*) versucht, genau wie auch SCAN und cSCAN, die Bewegung des Plattenarmes zu minimieren. Hier wird jedoch, ausgehend von der aktuellen Position des Plattenarmes, der als nächstes zu erreichende Auftrag aus der Warteliste angesprungen, unabhängig von der letzten Bewegungsrichtung des Armes.

  Bei den für diese Arbeit durchgeführten Versuchen mit zufälligen Lasten verhielt sich SSTF sehr ähnlich zu SCAN.

[2] Das scheint das heute in der Praxis übliche Verfahren zu sein (siehe auch [Poh02]).

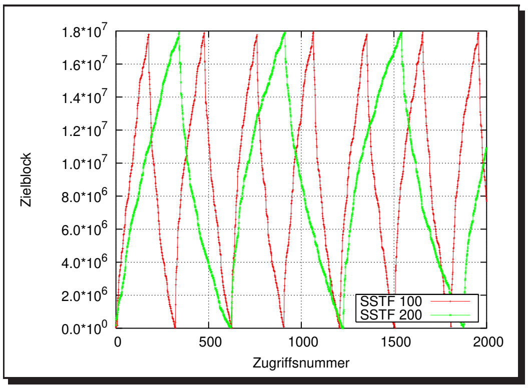


Abbildung 3.3: SSTF verhält sich bei zufälligen Zugriffsmustern ähnlich wie SCAN

- Beim **SATF**-Algorithmus (*shortest access time first*) wird versucht die Plattenauslastung weiter zu erhöhen, indem ein weiterer Leerlaufanteil minimiert wird — die Rotationsverzögerung. Wenn man Informationen über die genaue geometrische Position der einzelnen Sektoren auf der Festplatte sowie ein Verhaltensmodell besitzt, kann man versuchen die Ausführungszeit eines Plattenauftrages vorherzusagen. Ähnlich wie bei SSTF wird bei SATF die Ausführungszeit zum nächsten Auftrag minimiert und entsprechend diesem Kriterium ein Auftrag aus der Warteschlange ausgewählt.

  Gegenüber SCAN und SSTF ist bei SATF eine noch höhere Auslastung der Platte zu erwarten, da ein weiterer Leerlaufparameter minimiert wird. Andererseits wird für die Realisierung dieses Algorithmus deutlich mehr Wissen über die Platte benötigt, als bei irgendeinem der bisher genannten Algorithmen. Die geometrische Lage der einzelnen Sektoren auf der Platte, die Seek-Funktion, die Rotationsgeschwindigkeit und diverse weitere Parameter müssen teilweise vorher und teilweise dynamisch ermittelt werden.

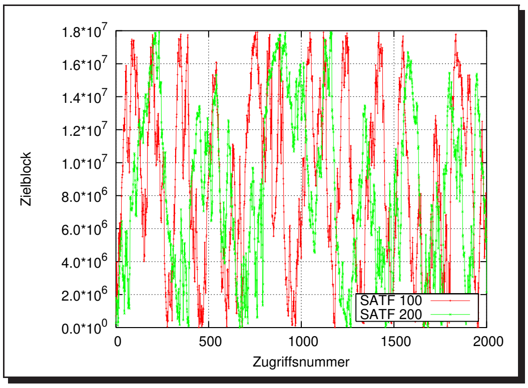


Abbildung 3.4: SATF lässt ansatzweise SCAN-ähnliche Muster erkennen, die aber stark von der Rotationszeitminimierung überlagert werden

- Eine weitere Leistungssteigerung gegenüber SATF lässt der **OAT**-Algorithmus (*optimal access time*) erwarten. Hier wird nicht nur eine Schedulingentscheidung für den nächsten

Plattenauftrag getroffen, sondern die optimale Bearbeitungsfolge für die ganze Warteschlange wird vorberechnet. JACOBSON und WILKES erwähnen aber in [JW91], dass für die Bestimmung der Reihenfolge keine schnellen Algorithmen existieren. Die optimale Reihenfolge lässt sich also nur für sehr kleine Auftragsmengen in akzeptabler Zeit ermitteln. Des Weiteren ist es sehr wahrscheinlich, dass während eine vorberechnete Folge abgearbeitet wird, neue Aufträge in die Warteschlange kommen und sich damit die Grundmenge, aus der eine Reihenfolge zu bestimmen ist, ständig ändert.

Zur Lösung von schwierigen Problemen, werden oftmals *“greedy”* (gierig) Algorithmen eingesetzt. Vertreter dieser Klasse optimieren nur lokal und sind deswegen relativ einfach zu berechnen, finden aber andererseits nicht immer das globale Optimum.

SATF und SSTF sind Vertreter der *“greedy”*-Klasse, da sie immer nur die Zeit zwischen zwei Aufträgen minimieren, also lokal entscheiden.

JACOBSON und WILKES erwähnen in [JW91], dass SATF eine sehr gute Annäherung an OAT darstellt, und betrachten OAT eher als theoretische Obergrenze.

Auch der besondere Umstand, dass nicht alle bereiten Aufträge in der Warteschlange auch auszuführen sind, lassen den OAT-Algorithmus hier ungeeignet erscheinen. Diese Situation kann in DROPS zum Beispiel durch die Verwendung eines Stromes der Klasse 2 (siehe Abschnitt 2.3.1) entstehen, für den nur eine 70-prozentige Erfüllung seiner Auftragsmenge eingeplant wurde. Die restlichen 30% der Aufträge werden am jedem Periodenende verworfen. Es ist aber vorher nicht klar, welche Aufträge zu diesen 30% gehören werden.

3.2.1 Auswahl

Aus der Menge der vorgestellten realisierbaren Schedulingalgorithmen ist mit SATF theoretisch die beste Auslastung der Platte zu erwarten.

Schon in Abschnitt 2.4.4 wurde erwähnt, dass Untersuchungsergebnisse in der Literatur ([HC02]) eine Implementierung dieses Algorithmus außerhalb der Platte (also zum Beispiel im Plattentreiber) als unmöglich beziehungsweise als nicht sinnvoll erscheinen lassen. Die im Folgenden beschriebenen selbst durchgeführte Voruntersuchungen ergaben ein anderes Bild (siehe Abschnitt 3.2.5).

Deshalb wurde hier dieser Algorithmus zur Implementierung ausgewählt. In der Praxis muss sich zeigen, ob es möglich ist ein gutes Plattenmodell zu erstellen, ob es sich mit vertretbarem Ressourcenaufwand implementieren lässt und ob notwendige Daten für das Modell schnell genug erfasst und verarbeitet werden können.

Ein weiteres Phänomen, welches neben der Plattenauslastung und damit der gelieferten Bandbreite betrachtet werden sollte, ist die maximale Bearbeitungszeit eines Auftrages. Bestimmte

Schedulingalgorithmen (zum Beispiel SCAN) verhindern das Verhungern einzelner Aufträge an sich schon. Bei SSTF ist theoretisch die Möglichkeit gegeben, dass bestimmte Aufträge niemals ausgeführt werden, da sich neu in die Warteschlange eingereihte Aufträge näher an der aktuellen Plattenarmposition befinden und somit immer bevorzugt werden.

Für Echtzeitaufträge in der DROPS-Architektur ist die Lösung des Problems nicht relevant, da Verhungern implizit durch die Periodenaufteilung verhindert wird. Die Lösung für Nicht-Echtzeitaufträge wird in dieser Arbeit nicht weiter betrachtet. Ansatzpunkte dazu werden nochmal in Kapitel 6 diskutiert.

In Abschnitt 2.4.1 wurden bereits Möglichkeiten erwähnt, vorhandene Algorithmen so zu modifizieren, dass Verhungern nicht mehr auftreten kann. Selbst diese Ansätze können aber keine quantifizierbaren Zeitgarantien geben und damit die Grundlage für den Echtzeitbetrieb innerhalb einer Periode schaffen. Für das Scheduling innerhalb der Klasse der Nicht-Echtzeitaufträge sind sie in DROPS eventuell sinnvoll einsetztbar.

3.2.2 Plattenmodell

Wie schon weiter oben angedeutet wurde, ist für die Vorhersage der Plattenauftragsbearbeitungszeiten ein mathematisches Modell notwendig. Hier wird ein Modell zur Vorhersage von einzelnen Auftragsbearbeitungszeiten aus [Poh02] als Ausgangspunkt genutzt:

$$\boxed{t_{request} = t_{seek} + t_{rotation} + m * t_{sector-rotation} + t_{overhead}} \tag{3.1}$$

Funktion 3.1 beschreibt die Ausführungszeit eines Plattenauftrages, der aus einzelnen Zeitanteilen besteht. Auf dieser Grundlage wird jetzt ein Modell entwickelt, welches die Zeit, die zwischen zwei Plattenaufträgen vergeht, modelliert. Das ist die für den SATF-Scheduler zu minimierende Zeit. Die beiden Aufträge sind zum Zeitpunkt einer Schedulingentscheidung der aktuell abgeschlossene und der als nächstes auszuwählende Auftrag:

$$\boxed{t_{between} = t_{seek}(\delta_{cylinder}) + t_{overhead} + t_{rotation}\big((\delta_{angle}, t_{overhead}, t_{seek}(\delta_{cylinder})\big)} \tag{3.2}$$

Dabei haben die einzelnen Summanden folgende Bedeutung:

- $t_{seek}(\delta_{cylinder})$ beschreibt die Zeit, die für die Seek-Bewegung zwischen dem Zylinder des aktuellen Plattenauftrages und dem Zielzylinder notwendig ist. Diese Größe hängt von der Entfernung der beiden Zylinder voneinander ($\delta_{cylinder}$) ab. Einzelne Plattenaufträge können sich auch über Zylindergrenzen hinweg erstrecken. In diesem Fall ist für den aktuellen Auftrag der Zylinder des letzten und für den nächsten Auftrag der Zylinder des ersten Sektors zu verwenden.

- $t_{overhead}$ steht für den Overhead, der zwischen dem aktuellen und dem nächsten Plattenauftrag auftritt.

- $t_{rotation}\big((\delta_{angle}, t_{overhead}, t_{seek}(\delta_{cylinder})\big)$ modelliert die Rotationszeit, die der Plattenkopf auf den Zielsektor warten muss, nachdem die Kopfpositionierung $t_{seek}(\delta_{cylinder})$ erfolgreich durchgeführt wurde und $t_{overhead}$ Zeit verstrichen ist. Dieser Parameter liegt zwischen 0 und einer vollständigen Rotation der Platte. Auch dieser Parameter ist nicht konstant, sondern hängt von der Winkeldifferenz (δ_{angle}) der beteiligten Sektoren, dem Overhead ($t_{overhead}$) und der Seek-Zeit ($t_{seek}(\delta_{cylinder})$) ab. Auch bei der Winkeldifferenz ist der Fall von Aufträgen mit mehr als einem Sektor besonders zu beachten. Hier ist wieder vom aktuellen der letzte und vom folgenden Auftrag der erste Sektor für die Winkelberechnung heranzuziehen.

Das Modell berücksichtigt nicht den Unterschied zwischen Kopfumschaltzeit[3] und Zylinderumschaltzeit[4], da sich in [Poh02] gezeigt hat, dass sich diese Zeiten bei modernen Platten nicht mehr signifikant unterscheiden.

Der Winkel und die Spur vom gerade abgeschlossenen Auftrag bilden zusammen die Kopfposition. Im nächsten Abschnitt wird die Bestimmung derselben diskutiert.

3.2.3 Bestimmung der Kopfposition

Festplatten bieten nach außen hin keine Schnittstelle an, um die aktuelle Kopfposition abzufragen.

Ein Ansatz, den aktuellen Winkel des Plattenkopfes zu bestimmen, ist die Berechnung aus der aktuellen Zeit. Das kann bei Festplatten funktionieren, da die Rotationsgeschwindigkeit hier relativ konstant ist. Dazu werden nur die Geschwindigkeit, ein genauer, aktueller Zeitstempel und eine Referenzmessung mit bekanntem Winkel benötigt. Dieser Ansatz wird in [HC02, LSG02] beschrieben. Der Nachteil dieser Methode ist, dass ab und zu zusätzliche Plattenaufträge nur zur Kalibrierung des Plattenmodells benötigt werden.

Wenn man über vollständige Geometrieinformationen der Platte verfügt und somit den Winkel und die Spur zu jedem Sektor ausrechnen kann, ist es möglich, jeden einzelnen Plattenauftrag dazu zu benutzen, die Kopfposition im Plattenmodell zu aktualisieren. Voraussetzung dafür ist, dass die Zeitdauer zwischen dem Zeitpunkt, an dem der Plattenkopf mit dem letzten Sektor des Auftrages fertig ist, und dem Zeitpunkt, zu dem der Interrupt vom Plattentreiber behandelt wird, konstant und messbar ist.

Das Histogramm in Abbildung 3.5 verdeutlicht, dass diese Voraussetzung in der Praxis

[3] Tritt bei Spurwechsel im selber Zylinder auf.
[4] Tritt bei Spurwechsel zwischen benachbarten Zylindern auf.

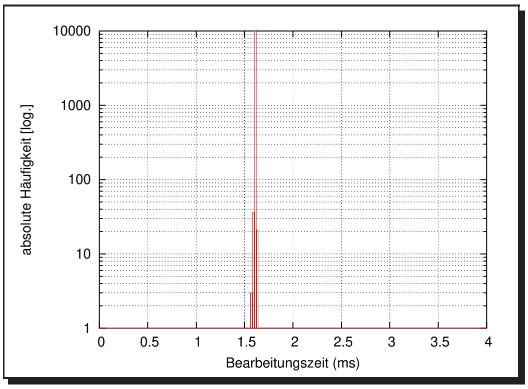


Abbildung 3.5: Histogramm mit absoluten Häufigkeiten über die Bearbeitungszeit von einem Zugriffspaar von Sektor 1.000 zu Sektor 10.000, 10.000 Wiederholungen (Platte: IBM IC35L018UWPR15-0)

durchaus erfüllt sein kann. Es sind fast keine Streuungen der Bearbeitungszeiten zu erkennen. Zugriffspaare über andere Entfernungen sehen ähnlich stark gruppiert aus.

3.2.4 Plattendatenbank

Die Ermittlung der notwendigen Plattenparameter baut auf den Ergebnissen aus [Poh02] auf. Die für SATF notwendigen Plattenparameter umfassen:

- Rotationsdauer der Platte,
- Spurgrenzen,
- geometrische Position aller Sektoren (Spurnummer, Winkel gegenüber einem Referenzpunkt — zum Beispiel Sektor 0),
- Seek-Funktion und
- Overhead-Zeiten für verschiedene Auftragstypen.

Für den Einsatz mit SCSI-Platten war außerdem eine Portierung der Extraktionssoftware erforderlich. Dazu war eine Anpassung des Linux-Kern-Patches und eine Reihe von weiteren kleinen Eingriffen notwendig. Insbesondere für TCQ war eine Anpassung zu entwerfen, die die vorhandene Software möglichst unverändert mit SCSI-Platten arbeiten lässt. Bei TCQ kann man sich nicht mehr darauf verlassen, dass nach dem Ereignis `Start_Auftrag_n` immer als nächstes `Ende_Auftrag_n` folgt.

Deshalb wurde ein kleiner Filter geschrieben, der die Ereignisse wieder in eine der Software verständliche Reihenfolge bringt. Auf die Genauigkeit der Zeitmessungen hat das keine negativen Auswirkungen, da jedes Ereignis schon bei seiner Erzeugung im Plattentreiber einen

Zeitstempel bekommt. Die Ergebnisse des Filters sind zum Beispiel über eine *"named pipe"* zugänglich und der Einsatz ist damit für die Extraktionssoftware vollkommen transparent.

Die Ermittlung der Plattenparameter basiert auf empirischen Messungen. Damit eine akzeptable Genauigkeit erreicht wird, müssen viele Parameter über Durchschnittswerte von sehr vielen Durchläufen bestimmt werden. Besonders die Ermittlung der Seek-Kurve und die der geometrische Positionen der Sektoren ist sehr zeitaufwendig.

Für die Repräsentation der Seek-Kurve werden viele Stützpunkte ermittelt. Im Bereich für kurze Sprünge liegen diese Stützpunkte dicht, für größere Sprungdistanzen werden weniger Punkte benutzt. Die Berechnung der Seek-Zeit für das Modell geschieht im laufenden Betrieb durch lineare Interpolation zwischen den umgebenden Stützpunkten.

3.2.5 Voruntersuchung zur Implementierung

Um vor einer aufwendigen Implementierung abschätzen zu können, ob in der Praxis überhaupt ein Gewinn von einem SATF-Scheduler zu erwarten ist, wurde ein Simulator als normaler Linux-Prozess auf Basis des in Abschnitt 3.2.2 entworfenen Modells implementiert. In diesem Simulator wurden verschiedene Algorithmen (FIFO, cSCAN, SSTF und SATF) realisiert und mit verschiedenen Zugriffsmustern getestet.

Der Simulator benötigt als Eingangsinformation eine Lastbeschreibung, die zum Beispiel in Form eines Logfiles einer realen Last vorliegen kann. Zu Beginn eines Simulationslaufes wird eine Warteschlange mit den ersten Aufträgen aus der Lastbeschreibung gefüllt. Auf dieser Warteschlange arbeiten die im Simulator implementierten Schedulingalgorithmen. Nachdem der zu simulierende Algorithmus einen nächsten Auftrag ausgewählt hat, wird dieser aus der Warteschlange entfernt und mittels eines *"raw-devices"*[5] (vgl. [Gil]) an die Platte weitergereicht. Daraufhin wird die Warteschlange wieder aufgefüllt. Das Zeitverhalten der Platte ist somit nicht Bestandteil der Simulation, sondern wird echt gemessen. Vielmehr wird eine Softwareumgebung für Scheduler bereitgestellt.

Schon in diesem Stadium zeigte sich, dass die Genauigkeit der extrahierten Plattenparameter ausreicht, um eine merkbare Leistungssteigerung gegenüber herkömmlichen Algorithmen zu erreichen. Die Messungen erfolgten mit IDE- und mit SCSI-Platten.

Die Histogramme über die Bearbeitungszeiten der verschiedenen Schedulingalgorithmen in Abbildung 3.6 verdeutlichen das oben dargelegte.

[5] Dadurch wird der Linux-Dateisystem-Cache umgangen.

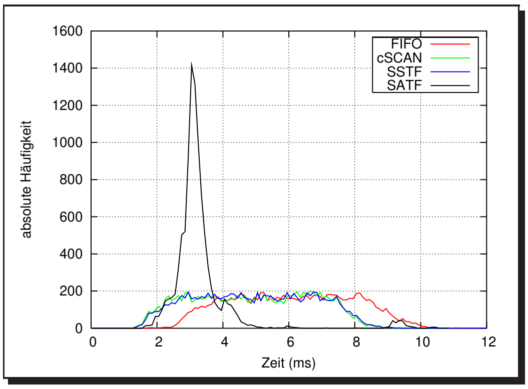


Abbildung 3.6: Dargestellt sind die ersten Simulationsergebnisse unter Linux — ein Histogramm über die Bearbeitungszeit von Plattenaufträgen für verschiedene Schedulingalgorithmen bei einer zufälligen Last. SATF setzt sich deutlich ab.

3.3 Anpassung an Echtzeitsysteme

Zu Beginn des Kapitels wurde bereits kurz angesprochen, dass Echtzeitgarantien nicht ohne weiteres durch eine besondere Konstruktion des Schedulers zu erreichen sind. Deshalb wird ein anderer Ansatz gewählt.

Die bisherige Implementierung des Plattenschedulers in DROPS — beschrieben in Abschnitt 2.3 — kann Echtzeitgarantien geben und erreicht das im Wesentlichen durch ein FIFO-Scheduling innerhalb der Klassen.

Der erste Schritt zur Verbesserung der Plattenleistung sollte die Ersetzung des FIFO-Algorithmus innerhalb der drei Auftragsklassen durch einen leistungsfähigen Scheduler sein. Dabei können immer noch die Echtzeitgarantien eingehalten werden.

Auf die Implementierung dieses Zwischenschrittes wurde verzichtet. Stattdessen wird ein Verfahren entwickelt, welches auch die Klassentrennung aufhebt. Dieses Verfahren erzeugt ein dynamisches Fenster auf der Menge der bereiten Aufträge und generiert dadurch eine Untermenge, aus der der Scheduler wählen kann.

Dabei sind zwei Bedingungen einzuhalten:

- Das Fenster muss möglichst groß sein, damit der Scheduler eine große Auswahl hat und somit eine hohe Plattenauslastung erreicht wird.
- Das Fenster muss so gewählt werden, dass bei Auswahl jedes beliebigen Auftrages alle bisher geltenden Echtzeitgarantien erhalten bleiben. So würde sich, zum Beispiel in einer Situation kurz vor Periodenende, wenn noch ein Echtzeitauftrag und viele Nicht-Echtzeitaufträge in der Warteschlange stehen, das Fenster nur über den Echtzeitauftrag erstrecken, da die Ausführung eines Nicht-Echtzeitauftrages die Verletzung der Zeitschranke des Echtzeitauftrages zur Folge hätte.

Die Entscheidung, ob bestimmte Aufträge zu einem Zeitpunkt im Fenster liegen, ist nicht für jeden Auftrag einzeln zu treffen, sondern nur einmal pro Auftragsstrom. Dabei wird unterschieden, ob ein Strom seine reservierte Zeit schon verbraucht hat oder nicht. Das dynamische Fenster wird für jede Schedulingentscheidung neu aufgebaut.

Im Folgenden ist der Algorithmus zur Erzeugung des Fensters angegeben:

```
window.empty()
period    = get_actual_period()
time_left = period.time_left()
wc_time   = disk_model.wc_time

// Worst-Case Ströme
for each wc_stream in period
   window.add(wc_stream)
   time_left -= wc_time * wc_stream.length

// normale Echtzeit-Ströme
if (time_left >= wc_time)
   for each rt_stream in period
      if (rt_stream.time_left > 0)
         window.add(rt_stream)
         time_left -= rt_stream.time_left

// Nicht-Echtzeit-Strom
if (time_left >= wc_time)
   window.add(nrt_stream)
```

Die Berechnung des Fensters (`window`) ist in drei Schritte aufgeteilt. Im Algorithmus wird die Variable `time_left`, die mit der Restzeit der aktuellen Periode initialisiert wird, an verschiedenen Stellen verkleinert und als Bedingung für die Ausführung einzelner Schritte benutzt.

Im ersten Schritt werden alle Worst-Case-Ströme in das Fenster aufgenommen, da sie immer zu 100% zu erfüllen sind. Aufträge dieser Klasse sind mit der Worst-Case-Ausführungszeit (`wc_time`) einzuplanen. Somit wird entsprechend der jeweiligen Stromgröße der Wert von `time_left` abgezogen.

Verbleibt nach dem ersten Schritt genug Zeit (`time_left >= wc_time`), werden *alle* normalen Echtzeitströme, deren reservierte Zeit noch nicht verbraucht ist, in das Fenster aufgenommen. Für diese Ströme wird ihre restliche Reservierungszeit von `time_left` abgezogen.

Wenn nach dem zweiten Schritt immer noch Zeit in der Periode übrig ist, so werden Nicht-Echtzeitströme betrachtet. Als Grenzwert für die Aufnahme dieser Aufträge wurde wiederum die Worst-Case-Ausführungszeit zugrunde gelegt. Nähere Unterscheidungen zwischen Strömen dieser Klasse müssen nicht getroffen werden, da alle entsprechenden Aufträge zu einem Strom gehören.

Denkbar wäre ein vierter Schritt, der wiederum nur bei vorhandener Restzeit durchlaufen wird. In diesem Schritt könnten alle normalen Echtzeitströme mit aufgebrauchten Reservierungszeiten und nicht bearbeiteten Auftragen in das Fenster aufgenommen werden. So könnte nicht benötigte Plattenbandbreite qualitätssteigernd eingesetzt werden.

4 Implementierung

In diesem Kapitel sind Besonderheiten der Implementierung beschrieben. Zuerst werden einige Werkzeuge näher beschrieben — der Schedulingsimulator unter Linux und die Programme zur Ermittlung der Modellparameter für SATF. Danach wird die grundlegende Struktur der Schedulingalgorithmen in dieser Arbeit näher erläutert, gefolgt von der Beschreibung der einzelnen Algorithmen. Den Schwerpunkt und gleichzeitig Abschluss dieses Kapitels bildet dabei SATF.

4.1 Hilfsprogramme

4.1.1 Simulator

Der schon im Entwurfskapitel 3.2.5 beschriebene Schedulingsimulator wurde vor der Implementierung in DROPS für vielfältige Experimente benutzt. Beispielsweise wurden die Auswirkungen der Auftragsgröße auf den optimalen Overhead-Parameter bei IDE-Platten gemessen — eine Abhängigkeit, die bei SCSI-Platten nicht nachgewiesen werden konnte. In Abschnitt 4.2.6.1 wird diese Beobachtung genauer erläutert. Des Weiteren zeichnete sich ab, dass SATF relativ empfindlich auf Ungenauigkeiten in der Seek-Kurve reagiert. Im Ergebnis der durchgeführten Experimente wurde die Extraktionssoftware für die Plattendatenbank schrittweise angepasst und verbessert.

Um abschätzen zu können, ob sich der Rechenzeitbedarf des im Simulator implementierten SATF-Algorithmus von aktuellen PC decken lässt, wurde ein Offline-Schedule eines Zugriffsmusters mit einem Online-Schedule des selben Musters verglichen. Für das Offline-Scheduling wurde die Reihenfolge der einzelnen Aufträge komplett vor der Ausführung berechnet. Zur Laufzeit wurde nur die vorher ausgerechnete Reihenfolge abgespielt. Im Gegensatz dazu werden beim Online-Scheduling alle Berechnungen vollständig zur Laufzeit ausgeführt.

Die Auftragsreihenfolgen, die von beiden Schedulingmethoden bestimmt wurden, waren identisch. Das liegt daran, dass SATF keine Zeitstempel, sondern nur das mathematische Plattenmodell benutzt, um die Reihenfolge vorherzubestimmen. Die beiden Durchläufe unterschieden sich also nur minimal in den Zeiten zwischen dem Starten der einzelnen Aufträge, da hier beim Online-Scheduling Berechnungen anfielen.

Die resultierenden Dichteverteilungen für 10.000 Einzelaufträge mit einer Warteschlangengröße von 100 zeigten auf einem 1 GHz Athlon-System keine erkennbaren Unterschiede. Die Berechnungen liefen also so schnell ab, dass kein erkennbarer Leistungsverlust durch das verzögerte Abschicken der Aufträge entstand.

Es kann also davon ausgegangen werden, dass sich SATF auf aktuellen Prozessoren implementieren lässt.

4.1.2 Erstellung der Plattendatenbank

Die vorliegende Arbeit baut teilweise auf einer früheren Arbeit ([Poh02]) auf, in der das Worst-Case-Verhalten von Festplatten untersucht wurde. Als Nebenprodukt entstanden Werkzeuge, die automatisch viele notwendige Parameter von IDE-Festplatten extrahieren können.

Im Rahmen dieser Arbeit wurden diese Werkzeuge erweitert, sodass:

- SCSI-Festplatten unterstützt werden,
- weitere notwendige Parameter (Rotationsoffsets der einzelnen Spuren, eine genaue Seek-Kurve) ermittelt werden,
- die nun erreichte Genauigkeit bei der Parameterextraktion den neuen Anforderungen gerecht wird und
- eine aktuellere Linux-Kern-Version (2.4.19) unterstützt wird.

Das Resultat der Parameterextraktion sind zwei Dateien, die die Geometrieinformationen, die Seek-Kurve und weitere Daten enthalten. Sie bilden die Plattendatenbank und werden bei der Initialisierung des SATF-Schedulers eingelesen, sodass der statische Teil des Plattenmodell daraus aufgebaut werden kann.

4.2 Schedulingalgorithmen

4.2.1 Struktur

Alle implementierten Algorithmen bestehen aus einem Plattenmodell und einer Reihe von Funktionen. Je nach Komplexität des Schedulingalgorithmus spiegelt das Plattenmodell den Zustand der Platte mehr oder weniger genau wieder. Hierbei kann man zwei Gruppen von Informationen unterscheiden:

- Die dynamischen Informationen werden nach jedem Auftrag aktualisiert. Sie modellieren meist in irgendeiner Form die aktuelle Position des Plattenarmes. Bei cSCAN und SSTF geschieht das zum Beispiel in Form der logischen Blockadresse des letzten Auftrages.

- Statische Informationen müssen nur bei der Initialisierung des Schedulers berechnet werden und bleiben dann konstant. Zu dieser Gruppe gehört beispielsweise die Plattengröße, wie sie bei cSCAN für ein schnelles Durchsuchen aller Aufträge benötigt wird, oder die Seek-Funktion, die bei SATF zur Optimierung der Zugriffszeit verwendet wird.

Die beiden Extrema bei der Modellkomplexität sind hier FIFO und SATF. Für den FIFO-Algorithmus werden überhaupt keine Informationen über die Platte benötigt, das Modell ist leer. Der SATF-Algorithmus benötigt in der implementierten Form das größte Modell.

Die entscheidende Funktion, die von allen Algorithmen implementiert werden muss, ist `get_next_request()`. Sie wird vom Plattentreiber aufgerufen, um den nächsten zu bearbeitenden Auftrag zu ermitteln. In dieser Funktion wird zuerst die Menge der zu schedulenden Aufträge aus der Menge der bereiten Aufträge bestimmt. Das kann im Prinzip für jeden Schedulingalgorithmus gleich geschehen und ist deshalb in eine Funktion ausgelagert (siehe auch Abschnitt 4.2.2). Als nächstes wird in Abhängigkeit vom Plattenmodell und dem konkreten Algorithmus ein bestimmter Auftrag ausgewählt. Dieser Auftrag wird aus der Liste der bereiten Aufträge entfernt und als Ergebnis der Funktion zurückgeliefert. Zusätzlich werden die dynamischen Informationen des Plattenmodells aktualisiert.

Die grundlegende Struktur der neuimplementierten Algorithmen entspricht damit folgendem Muster:

```
requests = compute_schedulable_requests
for each request in requests do
   choose best requests according to
   disc_model and algorithm
update_disc_model(best)
dequeue(best)
return best
```

Die weiterhin notwendigen Funktionen dienen der Initialisierung und der Freigabe des jeweils benutzten Plattenmodells. Durch diese generische Struktur kann das benutzte Schedulingverfahren zur Laufzeit umgeschaltet werden.

4.2.2 Berechnung des dynamischen Fensters

Der in Abschnitt 3.3 beschriebene Algorithmus wurde implementiert. Gegenüber den zwei im Entwurfskapitel genannten Bedingungen hat sich die Ablaufgeschwindigkeit dieses Algorithmus als kritisch erwiesen, da er oft und in zeitkritischen Bereichen eingesetzt wird. Die Tatsache, dass nicht für jeden Auftrag eine Entscheidung zu treffen ist, sondern nur eine pro Strom, hat sich in diesem Zusammenhang als sehr vorteilhaft erwiesen.

Die Speicherung der Rückgabewerte erfolgt in einem statischen Feld als Menge von Zeigern auf Ströme. Die Datenstrukturen der einzelnen Ströme liegen dabei schon fertig im Treiber vor, sodass zur Laufzeit keine Speicherallokation notwendig ist.

Um bestimmen zu können, welche Aufträge überhaupt ausgeführt werden dürfen, ist zuerst die Restzeitdauer der aktuellen Periode zu ermitteln. Diese ergibt sich aus der Differenz des Zeitpunktes des nächsten Periodenbeginns und dem aktuellen Zeitpunkt. Die Periodengrenzen sind an die Zeitquelle in der Kernel-Info-Page (KTIMER) gekoppelt. Diese Zeitquelle wird jedoch nur etwa einmal pro Millisekunde aktualisiert. Diese Genauigkeit erscheint nicht ausreichend, da die durchschnittliche Bearbeitungszeit eines Plattenauftrages bei nur 5 ms liegen kann (Platte: IC35L018UWPR15-0).

Alternativ bietet sich als Zeitquelle das TSC-Register[1] wegen seiner sehr guten Zeitauflösung an. Bis vor kurzem waren beide Zeitquellen nicht synchronisiert[2]. Deshalb musste, neben den Zeiten für die Periodengrenzen, auch die aktuelle Zeit aus dem KTIMER ermittelt werden. In praktischen Versuchen scheint dessen relativ grobes Aktualisierungsintervall aber noch auszureichen.

Probleme sind zu erwarten, wenn noch wesentlich schnellere Festplatten beziehungsweise kürzere Periodendauern (im Moment 500 ms) verwendet werden.

Eine Umstellung des gesamten SCSI-Treibers auf die Zeitquelle TSC wäre mit relativ vielen Eingriffen in den Quellcode verbunden gewesen. Gegen diesen Ansatz spricht auch die Tatsache, dass alle Programme, die mit dem SCSI-Treiber kommunizieren eine gemeinsame Zeitquelle brauchen. Ohne gemeinsame Zeitquelle ist die Verständigung auf Zeitschranken für Plattenaufträge unmöglich.

4.2.3 FIFO

Der FIFO-Algorithmus wurde im Rahmen dieser Arbeit nicht neu implementiert. Hier wird die bisherige Scheduler-Implementierung, die in Abschnitt 2.3 beschrieben ist, als FIFO bezeichnet. Dieser Algorithmus benutzt als einziger nicht die in Abschnitt 4.2.2 beschriebene Funktion um Auftragsuntermengen zu bilden[3]. Er dient hier im Wesentlichen als Basis für Leistungsvergleiche, aber auch als einfache Fallback-Implementierung.

[1] Das TSC-Register *(time stamp counter)* 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. Das Register zählt die Prozessortakte seit dem Starten des Rechners und stellt damit eine extrem feingranulare Zeitbasis zur Verfügung.

[2] Die aktuelle Version des L4/Fiasco-Kerns (12.02.2003) wurde auf Vorschlag des Autors um eine entsprechende Synchronisierungsoption erweitert.

[3] Er würde auch keinerlei Vorteil daraus ziehen können.

4.2.4 cSCAN

Für den cSCAN-Algorithmus muss als einzige statische Information im Plattenmodell die logische Plattengröße bekannt sein, um schnell entscheiden zu können, wann ein Rücksprung erforderlich ist. Für den dynamischen Teil des Plattenmodells ist nur die logische Adresse des zuletzt bearbeiteten Plattenauftrages zu speichern. Für den SCAN-Algorithmus, bei dem sich der Plattenarm in beide Richtungen bewegt, wäre im dynamischen Anteil des Plattenmodells noch die Information über die letzte Bewegungsrichtung abzulegen.

Annahme bei der Benutzung der logischen Adressen zur Positionsspeicherung ist, dass für die Platte lineares Mapping vorliegt.

Der prinzipielle Ablauf von cSCAN ist folgender:

```
requests = compute_schedulable_requests
last     = disc.model.last_request
for each request in requests do
   address = request.address
   if (address < last.address)
      address += disc.size
   if (address - last < best_diff || best == NULL)
      best_diff = address - last
      best      = request
update_disc_model(best)
dequeue(best)
return best
```

4.2.5 SSTF

Auch der SSTF-Algorithmus benötigt im Plattenmodell lediglich die Adresse des zuletzt ausgeführten Auftrages. Ein statischer Anteil ist nicht notwendig. In der im Folgenden angegebenen Implementierung wird von der Entfernung der logischen Adressen auf die physische Entfernung der Zielspuren geschlossen. Voraussetzung ist hier erneut lineares Mapping. Dieses Verfahren stellt eine Näherungslösung dar, da sich die physische Spurdistanz von zwei logisch gleich weit entfernten Adressen in Abhängigkeit der Plattenzone unterscheiden kann. In der Praxis funktioniert das Verfahren recht gut.

Hier ist der Aufbau wie folgt:

```
requests = compute_schedulable_requests
last     = disc.model.last_request
for each request in requests do
   address = request.address
   if (abs(address - last) < best_diff || best == NULL)
      best_diff = address - last
      best      = request
update_disc_model(best)
dequeue(best)
return best
```

4.2.6 SATF

Der SATF-Algorithmus versucht (siehe auch Abschnitt 3.2), gegenüber SSTF zusätzlich die Rotationsverzögerung zu minimieren. Deshalb kann hier nicht mehr mit logischen Adressen gearbeitet werden. Alle wesentlichen Positionen werden mit ihrem Winkel und ihrer Spurnummer gespeichert, also ihrer physischen Position auf der Platte. Es ist somit notwendig, logische Adressen in derartige Positionen umrechnen zu können. Das geschieht mit einer Reihe von Hilfsfunktionen und den Geometrieinformationen, die im statischen Teil des Plattenmodell abgelegt sind.

Damit stellt sich der Algorithmus wie folgt dar:

```
requests = compute_schedulable_requests
model    = disk.model
for each request in requests do
   address  = request.address
   angle    = get_angle_for_sector(model, address)
   track    = get_track_for_sector(model, address)
   overhead = choose_overhead(model.last.rw, request.rw)
   time = compute_request_time(angle, track, model, overhead)
   if (time < best_time || best == NULL)
      best_time = time
      best      = request
update_disc_model(best)
dequeue(best)
return best
```

Bei diesem Algorithmus ist die eigentliche Funktionalität in der Funktion `compute_request_time()` versteckt. Dort wird die voraussichtliche Zeit bis zum eigentlichen Beginn das Auftrages vorberechnet, also der gesamte zu minimierende Overhead[4]. Im Wesentlichen sind folgende Informationen notwendig:

- Zielspur,
- Zielwinkel,
- Startspur,
- Startwinkel,
- Zeit, um den Spurunterschied zu überwinden (Seek-Funktion), und
- der typische Overhead zwischen den zwei Auftragstypen (Start, Ziel).

In Abschnitt 2.1.2 wurde der zeitliche Ablauf eines Plattenauftrages dargestellt. Schon dort wurde deutlich, dass die Längen der einzelnen Phasen von der Auftragsart, dem Ziel und weiteren, zu untersuchenden Parametern abhängen. Deshalb müssen sie im Modell berücksichtigt werden.

4.2.6.1 Plattenmodell

Um den SATF-Scheduler so konstruieren zu können, dass nur der letzte Auftrag dazu benutzt wird, die aktuelle Position des Plattenarmes vorherzusagen, muss für jeden Auftrag der Ende-Overhead bekannt sein. Auch der Anfangs-Overhead für den als nächstes zu startenden Auftrag wird benötigt.

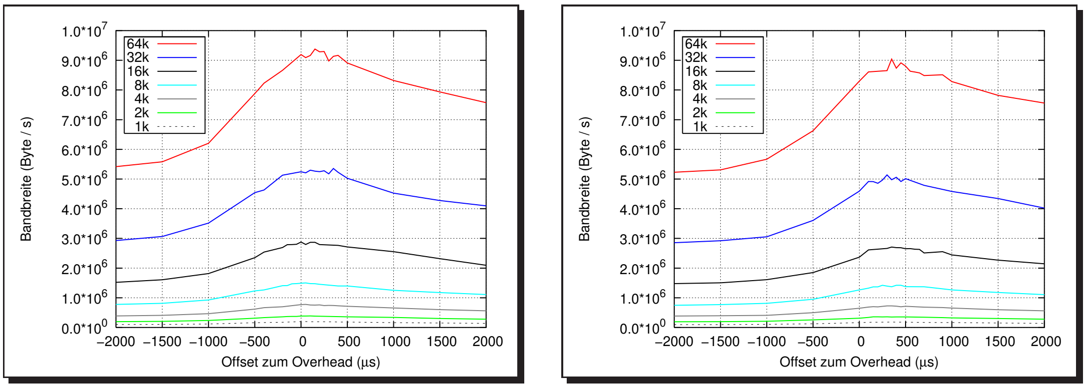
Gegenüber dem Plattenmodell 3.2 aus Abschnitt 3.2.2 hat es sich als notwendig erwiesen $t_{overhead}$ nicht als konstant anzunehmen, sondern mit dem aktuell abgeschlossenen und dem nächsten Auftragstyp zu parametrisieren. Das ist erforderlich, da die eben erwähnten Summanden, aus denen sich der Overhead zusammensetzt, nicht konstant sind:

$$\boxed{\begin{aligned} t_{between} = {} & t_{seek}\big(\delta_{cylinder}\big) + t_{overhead}\big(rw_{this}, rw_{next}\big) + \\ & t_{rotation}\big(\delta_{angle}, t_{overhead}(rw_{this}, rw_{next}), t_{seek}(\delta_{cylinder})\big) \end{aligned}} \tag{4.1}$$

Einige Eigenschaften von SCSI-Platten erleichtern hier die Modellierung. Wie schon in den Abbildungen 2.2 auf Seite 5 zu sehen war, finden bei Plattenzugriffen gleichzeitig Datenübertragung zwischen Rechner und Platte sowie Medienzugriffe der Platte statt. Deshalb ist anzunehmen, dass die einzelnen Overhead-Anteile unabhängig von der Größe des Plattenauftrages

[4] Das sind im Einzelnen der Auftragsende-Overhead des vorhergehenden, der Auftragsbeginn-Overhead und der Positionierungs-Overhead des auszuführenden Auftrages.

ausfallen, da immer nur die gleiche Restdatenmenge vor oder nach dem Plattenauftrag zu transferieren ist.



(a) Lesende Zugriffe

(b) Schreibende Zugriffe

Abbildung 4.1: Bandbreite in Abhängigkeit vom Offset zum Overhead-Parameter gemäß Modell 4.1 für verschiedene Plattenauftragsgrößen

Um diese Annahmen noch einmal praktisch zu überprüfen, wurde ein Benchmark geschrieben, der die Overhead-Summen im Modell variiert und die resultierende Leistung ermittelt. Die Graphen in Abbildung 4.1 zeigen die Ergebnisse dieses Benchmarks für Schreib- und Lesezugriffe mit jeweils verschiedenen Auftragsgrößen. Die Vermutung wird bestätigt, da die Maxima immer im gleichen Bereich liegen. Folglich muss die Auftragsgröße nicht im Modell berücksichtigt werden.

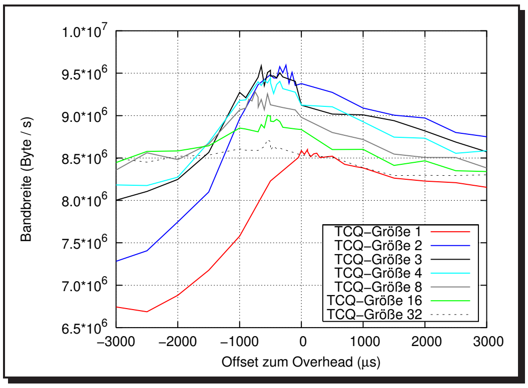
Voraussetzungen dafür sind: Der Übertragungsbus zwischen Platte und Kontroller muss schneller sein als die Dauertransferrate der Platte, und die Übertragung der Daten zwischen Platte und Kontroller muss parallel zu weiteren Medienzugriffen der Platte erfolgen. Die erste Bedingung ist praktisch relativ einfach durch den Einsatz eines schnellen Kontrollers zu erreichen. Die zweite Bedingungen ist, wie gezeigt wurde, bei modernen SCSI-Systemen ebenfalls gegeben. Versuche mit IDE-Platten ergaben hier das Gegenteil: Die kompletten Daten wurden hier vor beziehungsweise nach dem Auftrag übertragen. Bei einer Portierung des Schedulingalgorithmus und einer entsprechenden Anpassung des Modells für IDE muss das berücksichtigt werden.

Nach diesen Einschränkungen bleiben noch vier Parameter zu bestimmen, nämlich jeweils die Anfangs- und Ende-Overheadzeiten für Lese- und Schreibaufträge. Diese Parameter sind in der Praxis nicht ohne zusätzliche Hardware zu messen[5]. Bei genauer Betrachtung sind nicht unbedingt diese vier einzelnen Parameter notwendig, sondern nur jede Summenkombination von Aufträgen, die hintereinander ausgeführt werden können:

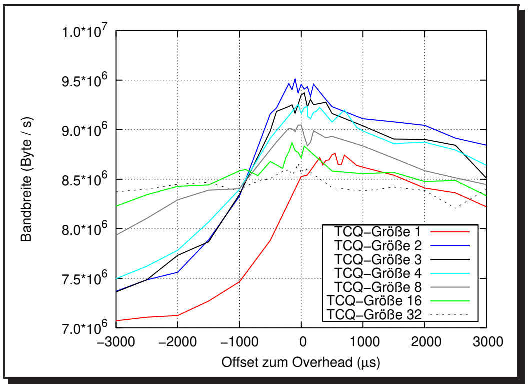
[5] Zur Softwaremessung bietet sich ein instrumentierter Treiber an. Als Messpunkte stehen dabei jeweils Auftragsstart und Auftragsende zur Verfügung. Die notwendigen Zwischenzeitpunkte, die zur Ermittlung der Overheadzeiten notwendig wären, stehen außerhalb der Festplatte nicht zur Verfügung.

1. *Leseauftrags-Endeoverhead* + *Leseauftrags-Anfangsoverhead*,
2. *Leseauftrags-Endeoverhead* + *Schreibauftrags-Anfangsoverhead*,
3. *Schreibauftrags-Endeoverhead* + *Leseauftrags-Anfangsoverhead* und
4. *Schreibauftrags-Endeoverhead* + *Schreibauftrags-Anfangsoverhead*.

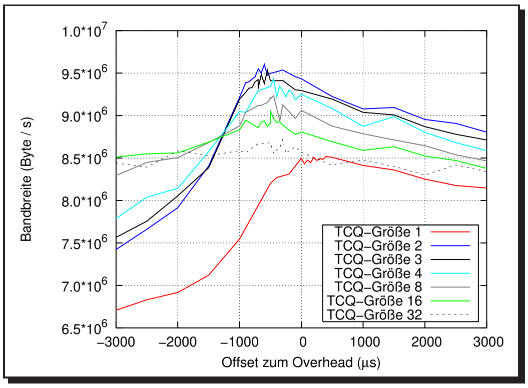
Diese vier Summen können experimentell bestimmt werden. Dazu wird eine gemischte Plattenauftragsserie (Lese- und Schreibzugriffe) an die Platte geschickt und gemäß dem mathematischen Modell 4.1 eingeplant, parallel dazu werden die einzelnen Overhead-Summen im Modell variiert. Die Maximumsuche kann für jede der Summen einzeln erfolgen, da sie unabhängig sind. Andererseits sind diese Summen *nicht* unabhängig von der verwendeten TCQ-Größe, da eine vergrößerte Warteschlange in der Platte einen Teil der Overhead-Summen kompensiert.



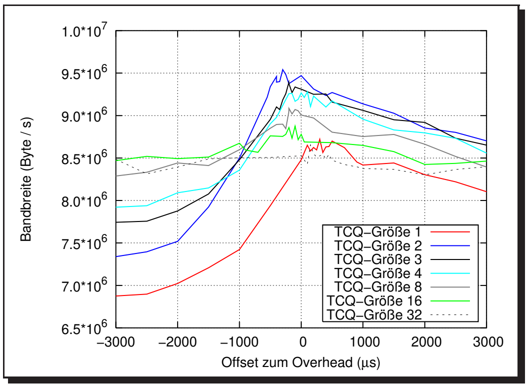
(a) Lese-Lese-Zugriffe



(b) Lese-Schreib-Zugriffe



(c) Schreib-Lese-Zugriffe



(d) Schreib-Schreib-Zugriffe

Abbildung 4.2: Bandbreite für verschiedene TCQ-Größen in Abhängigkeit vom Offset der Overhead-Summen (Platte: Seagate ST318438LW)

Dieser Zusammenhang ist in den Graphen in Abbildung 4.2 verdeutlicht. Die praktisch ermittelten Maxima für die einzelnen Summen bei verschiedenen TCQ-Größen werden nun in das

Modell eingesetzt und die resultierende Leistung kann gemessen werden. Das Ergebnis dieser Messung ist in Abbildung 4.3 dargestellt.

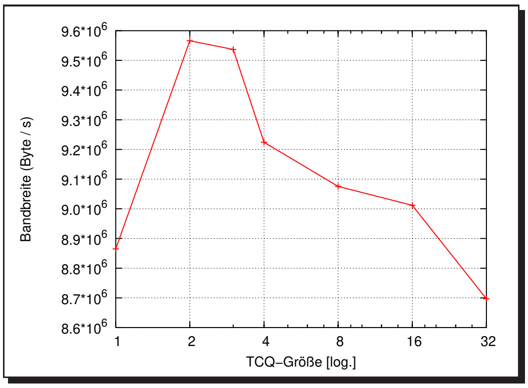
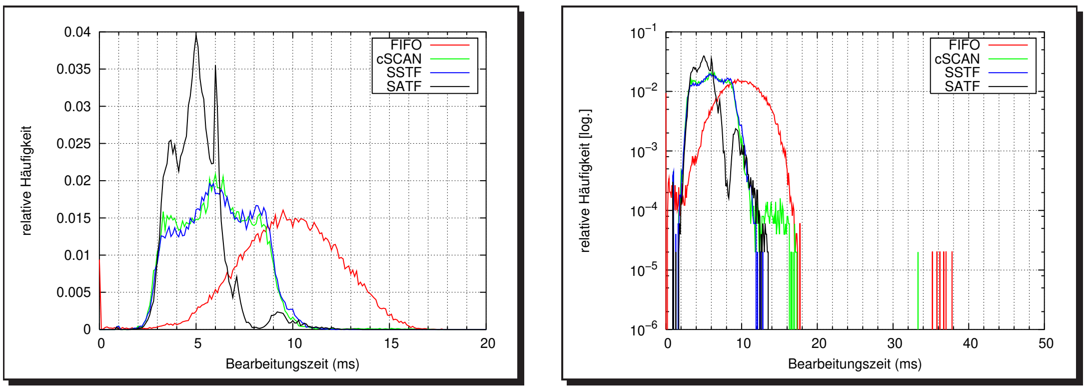


Abbildung 4.3: Resultierende Bandbreite nach Bestimmung der vier optimalen Overhead-Summen für verschiedene TCQ-Größen (Platte: Seagate ST318438LW)

Es scheint auch möglich, den Overhead-Zeitanteil an der Gesamtausführungszeit zu minimieren. Dazu kann *Tagged Command Queueing* verwendet werden. Wenn man darauf verzichtet, die Warteschlangengröße sehr groß zu wählen, sind auch keine negativen Auswirkungen auf die Echtzeitfähigkeit zu erwarten. Tatsächlich sollte eine Größe von zwei ausreichen, sodass immer gerade ein Kommando übertragen und eines ausgeführt wird. Dass diese Vermutung in der Praxis zutrifft, ist in Abbildung 4.3 gut erkennbar. Bei einer Warteschlangengröße von zwei kann die Platte auch keine Aufträge umsortieren, da niemals zwei bereite Aufträge gleichzeitig in der Warteschlange der Platte stehen. Wird eine längere Warteschlange zugelassen, fällt die Übertragungsleistung wieder. Das deutet daraufhin, dass die Platte die einzelnen Aufträge umsortiert und damit schlechtere Schedulingentscheidungen trifft als der hier benutzte Schedulingalgorithmus (SATF).



(a) Dichteverteilungen

(b) Worst-Case

Abbildung 4.4: Histogramm der Bearbeitungszeiten von Plattenaufträgen mit verschiedenen Schedulingalgorithmen (Platte: Seagate ST318406LW)

In Abbildung 4.4 ***(a)*** sind Histogramme der Bearbeitungszeiten für verschiedene Schedulingalgorithmen dargestellt. Das vermessene Muster besteht aus zufällig über die ganze Platte verteilten Zugriffen. Bei Verwendung von FIFO ergibt sich eine ungefähr normalverteilte Dichtefunktion, deren Zentrum bei etwa 10 ms liegt. Die Dichteverteilungen des cSCAN- und des SSTF-Algorithmus sind sehr ähnlich. Ihre Mittelwerte liegen etwa bei 6 ms. Diese Verwandtschaft war zu erwarten, da sich beide Algorithmen auch bei anderen Versuchen in dieser Arbeit gleichartig verhalten. Die Verwendung des SATF-Algorithmus resultiert in der kompaktesten Dichteverteilung. Begründen lässt sich das sehr gut durch die für diesen Algorithmus typische Minimierung der Rotationslatenz. Sie ist für einen Teil der Breite der Dichteverteilungen verantwortlich. Hier liegt der Mittelwert etwa bei 5 ms.

Abbildung 4.4 ***(b)*** stellt die selben Daten noch einmal logarithmisch skaliert dar. Zwischen 30 und 40 ms liegen die hier aufgetretenen Worst-Case-Zeiten. Interessanterweise gehören diese fast vollständig zu FIFO, und nur ein Zugriff aus diesem Bereich gehört zu cSCAN. Vermutlich wird durch die Schedulingalgorithmen nicht nur die durchschnittliche Bearbeitungszeit beeinflusst, sondern auch die Wahrscheinlichkeit, dass ein Worst-Case auftritt.

4.2.6.2 Laufzeit

Rotationssensitive Schedulingalgorithmen reagieren besonders empfindlich auf Verzögerungen beim Absenden von Aufträgen an die Platten, da sich der aktuelle Winkel des Plattenkopfes natürlich ständig ändert. Aus diesem Grund und weil es im Allgemeinen auch wichtig ist, dass nicht viel Rechenzeit in Treibern verbraucht wird, wurde bei der Implementierung besonders auf Ablaufgeschwindigkeit geachtet.

Die Geometrieumrechnung hat einen wesentlichen Anteil an den Laufzeitkosten. Die dabei eingesetzte binäre Separierung konnte durch Umsortierung der einzelnen Abbruchbedingungen leicht beschleunigt werden. Starke weitere Beschleunigung scheint nicht möglich, da die Routine nur noch ca. 1450 Prozessorzyklen benötigt[6]. Problematisch ist, dass sie sehr oft aufgerufen wird, nämlich zu jedem Schedulingzeitpunkt für jeden einzelnen Auftrag einmal. Hier bot sich intensives Caching der umgerechneten Werte an. Damit konnte eine starke Beschleunigung des Gesamtablaufs erreicht werden. Der Aufruf erfolgt nun nur noch einmal pro Auftrag, also im Regelfall zum ersten Schedulingzeitpunkt, nachdem der Auftrag in die Warteschlange eingereiht wurde.

Durch eine geschlossene Funktionsdarstellung der Seek-Kurve könnten noch einige Prozessorzyklen gewonnen werden. Die im Moment benutzte binäre Separierung zur Suche in einer Tabelle mit anschließender linearer Interpolation benötigt ca. 300 Takte.

Eigentlich sollte die Spurnummer zu einem gegebenen Sektor leicht auszurechnen sein, da die Spurgröße innerhalb einer Plattenzone konstant ist. Ungünstigerweise gibt es aber viele

[6]Die Messungen erfolgten auf dem in Abschnitt 5.1 beschriebenen Testrechner.

Unregelmäßigkeiten durch fehlerhafte Sektoren. Das lässt sich besonders gut in den Graphen der Zoneneinteilung der einzelnen Platten im Anhang A ablesen. Eventuell kann hier durch den Einsatz einer anderen Datenstruktur mit Defektlisten die Umrechnung weiter beschleunigt werden.

5 Bewertung

5.1 Messumgebung

Abbildung 5.1: Schnelle Festplatten brauchen gute Kühlung . . .

Der verwendete Testrechner, auf dem fast[1] alle Messwerte aufgenommen wurden, besitzt folgende Ausstattung:

Prozessor:	Intel Pentium Pro 200 MHz
SCSI-Controller:	“Tekram DC-390U2W Ultra-Wide” Kontroller (80 MB/s)
Betriebssystem:	DROPS mit L4/Fiasco
Speicher:	128 MB
Festplatte:	verschiedene (siehe Anhang A)

5.2 Admission

Die Admission (siehe auch Abschnitt 2.2.2) für das Modell der schwankungsbegrenzten Ströme (vgl. [HLR$^+$01]) benutzt als Eingangsdaten die Dichteverteilung der Bearbeitungszeiten und

[1] bis auf einige Simulationsergebnisse unter Linux für die Voruntersuchung

die Strombeschreibungen. Durch die Verbesserung des Schedulers traten dabei einige Probleme zu Tage, die im Folgenden diskutiert werden sollen.

5.2.1 Probleme

Die Admission beeinflusst die aktuelle Lastsituation sowohl direkt, indem bestimmte Ströme zugelassen werden oder nicht, als auch indirekt, indem für jeden Strom Zeitkonten festgelegt werden und damit implizit die Menge der abzuarbeitenden Aufträge.

Bei der bisherigen Implementierung des Plattenschedulers (FIFO) war die Dichteverteilung unabhängig von der aktuellen Lastsituation, da FIFO unabhängig vom Füllstand der Warteschlange entscheidet. Sobald ein Schedulingalgorithmus benutzt wird, der versucht, die Plattenauslastung zu optimieren (zum Beispiel cSCAN, SSTF oder SATF), und der deshalb vom Füllstand der Warteschlange abhängt, ist die Unabhängigkeit zwischen der Verteilung der Bearbeitungszeiten und der aktuellen Lastsituation nicht mehr gegeben. In Abbildung 5.2 ist der Zusammenhang zwischen Füllstand der Treiberwarteschlange und der gelieferten Bandbreite dargestellt. Hier zeigt sich nur FIFO unbeeinflusst, was sich aber auch in der schlechtesten Bandbreite äußert.

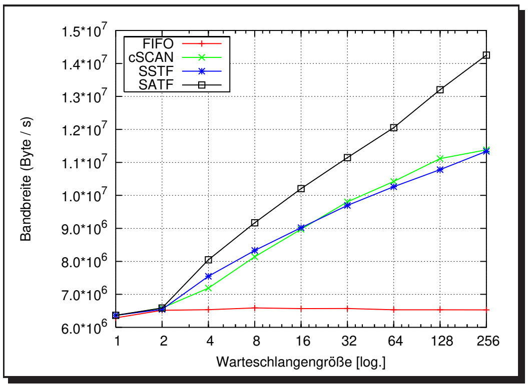


Abbildung 5.2: Dargestellt ist die Bandbreite, die das System mit unterschiedlichen Füllständen der Treiberwarteschlange und unterschiedlichen Schedulingalgorithmen liefern kann (Platte: Seagate ST318406LW)

Die Admission beeinflusst die Menge der bereiten Aufträge und modifiziert somit die Verteilung auf deren Basis sie selbst entscheidet und Zeitkonten ausrechnet.

Das führt zu mehreren Problemen:

- Um die Admission durchführen zu können, wird eine Verteilung benötigt, die erst durch die Admission festgelegt wird.
- Wenn zur Laufzeit bei einer eingeplanten Strommenge eine Teillast ausfällt — also die Gesamtlast des Systems reduziert wird — so steigt die mittlere Bearbeitungszeit für die

einzelnen Aufträge der restliche Ströme an. Diese Ströme verbrauchen also für eine kleinere Menge von Aufträgen ihr vorher ausgerechnetes Zeitkonto. Damit sinkt ihre Qualität, die als Anteil der erfüllten Aufträge definiert ist. Es tritt also folgendes Paradoxon ein: Sinkt die Systemlast, so sinkt auch die Qualität der einzelnen Ströme.

5.2.2 Lösungsvorschläge

Für das erste Problem bietet sich aufgrund der Abhängigkeit der Berechnung von ihren Ausgangswerten ein iteratives Verfahren an.

Dazu wurde folgender Versuch unternommen:

1. Die Admission wird mit einer Anfangsverteilung und der Strombeschreibung gestartet.
2. Mit dem Ergebnis des vorhergehenden Schrittes (berechnete Zeitkonten) wird das System gestartet. Dabei wird wieder die Verteilung der Bearbeitungszeiten aufgezeichnet.
3. Mit der neu gewonnenen Verteilung wird erneut die Admission gestartet.
4. Das nun gewonnene Ergebnis wird wieder mit dem realen System ausgeführt, wobei wieder die Verteilung aufgezeichnet wird.
5. Es wird solange erneut bei Punkt 3 begonnen, bis die Ergebnisse des praktischen Versuches nahe genug am gewünschten Ziel liegen.

Bei der praktischen Durchführung trat bei den ersten drei Iterationen tatsächlich eine Konvergenz zum Ziel hin auf. Weitere Wiederholungen brachten jedoch keine weitere Annährung, obwohl die Zielqualität immer noch deutlich unterschritten wurde.

Eine manuelle Korrektur der von der Admission ermittelten Zeitkonten zeigt, dass es durchaus eine stabile Lösung gibt. Auch bei praktischer Überprüfung dieser Werte wurde die Verteilung der Bearbeitungszeiten aufgenommen. Die Admission, angewandt auf die so gewonnene Verteilung, resultiert wieder in einer starken Unterdimensionierung der Zeitkonten, und zwar so, dass etwa der Konvergenzpunkt des iterativen Verfahrens erreicht wird.

Festzuhalten bleibt somit, dass das Verfahren prinzipiell konvergiert. Das gewünschte Ziel wird aber nicht genau erreicht.

Das Problem hierbei ist, dass durch die Vorauswahl der Aufträge die durchschnittlichen Warteschlangengrößen, bei denen Echtzeitaufträge und Nicht-Echtzeitaufträge ausgeführt werden, unterschiedlich sind. Daraus folgt, dass sich die Dichteverteilungen der Bearbeitungszeiten für Echtzeit- und Nicht-Echtzeitaufträge unterscheiden. Auch dieses Phänomen war bei der bisherigen Schedulerimplementierung (FIFO) nicht zu beobachten.

Als Lösung bietet es sich an, nur die Dichteverteilung der Echtzeitaufträge für die Admission zu verwenden. Bisher wurde die Gesamtverteilung des Systems genutzt. Damit stimmen die berechneten Zeitkonten viel genauer mit dem Optimum[2] überein. In Abbildung 5.3 wird ersichtlich, dass der Unterschied der Dichteverteilungen nicht sehr groß, aber in der Praxis durchaus bemerkbar ist. Bei Verwendung von TCQ-Größen über eins tritt eine leichte Überdimensionierung der Zeitkonten auf. Das wird im nächsten Abschnitt mit Zahlen belegt. Vermutlich lässt sich die Verteilung der Echtzeitaufträge von der der Nicht-Echtzeitaufträge nicht mehr so gut trennen wie bei der Verwendung einer TCQ-Größe von eins.

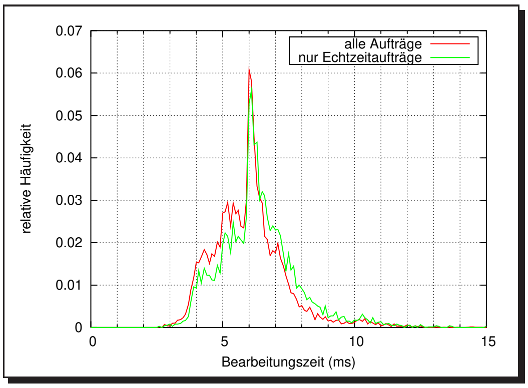


Abbildung 5.3: Dichteverteilungen von Plattenauftragsbearbeitungszeiten mit einer TCQ-Größe von 1 (Platte: Seagate ST318406LW)

Generell muss gesagt werden, dass sich das praktisch ausgeführte iterative Vorgehen für die Admission in einem normalen Echtzeitsystem verbietet, da von der Admission zur Laufzeit schnell Entscheidungen zu treffen und Zeitkontoberechnungen auszuführen sind. Diese Berechnungen fallen bei der Anmeldung neuer Ströme an. Damit kommt ein zeitaufwendiges iteratives Vorgehen nicht in Frage. Hier bieten sich nach kurzer Betrachtung folgende Lösungsansätze an:

- Die FIFO-Dichteverteilung wird für die Zeitkontoberechnung zugrunde gelegt, was eine starke Überdimensionierung und damit eine höhere Qualität der einzelnen Ströme zur Folge hat. Damit wird jedoch viel Bandbreite verschenkt.
- Es wird ein Korrekturverfahren für die Dichteverteilung entwickelt. Mit diesem Verfahren lässt sich die Auswirkung der Admission auf die Verteilung vorhersagen. Damit könnte die Admission das Ergebnis in einem Schritt liefern.
- Gegen ein iteratives Vorgehen spricht vor allem der Zeitbedarf. Mithilfe einer schnellen Simulation des verwendeten Schedulingverfahrens inkl. der Auftragsvorauswahl könnte der Zeitbedarf drastisch gesenkt werden. Das iterative Verfahren würde also nur die schnelle Simulation nutzen.

[2] Das Optimum ist hier für jeden Strom eine bestimmte Zeitkontogröße, sodass die Qualitätsanforderungen im praktischen Versuch genau erfüllt werden können.

Für das zweite Problem (eine Verringerung der Systemlast verschlechtert die Qualität der einzelnen Ströme) empfiehlt sich die in Abschnitt 3.3 vorgeschlagene Lösung. Bei der Berechnung der Untermenge der zu schedulenden Aufträge sollte eine vierte Stufe vorgesehen werden. In dieser Stufe werden für den Fall, dass noch genügend Restzeit in der Periode übrig ist, alle Ströme mit aufgebrauchtem Zeitkonto wieder in die Untermenge aufgenommen. Wenn die Systemlast reduziert wird, entsteht eine Situation, in der Restzeit in jeder Periode verfügbar ist. Sollten später andere Ströme neu von der Admission zugelassen werden, so sorgen diese wiederum für eine höhere Systemlast und kompensieren das Problem. Die Admissionberechnungen sollten also immer auf einer Volllastverteilung beruhen.

5.3 Ergebnisse und Bewertung

Generell ist es gelungen, einen Schedulingalgorithmus zu implementieren (SATF), der eine sehr gute Festplattenauslastung und damit eine hohe Leistung ermöglicht. Dieser Algorithmus bietet auch deutliche Leistungsvorteile gegenüber herkömmlichen Algorithmen wie cSCAN und SSTF.

Die konkreten Messwerte der einzelnen Festplatten sind in Anhang A zusammengefasst.

Des Weiteren konnte ein Mechanismus entwickelt werden, bei dem durch Vorauswahl von Plattenaufträgen die Echtzeitfähigkeit sichergestellt wird. Dieser Mechanismus ist unabhängig vom eingesetzten Schedulingalgorithmus, welcher deshalb beliebig ausgetauscht werden kann. Diese Vorauswahl schafft die Synthese zwischen der Benutzung eines leistungsfähigen Schedulers und der Einhaltung von Echtzeitschranken.

In Tabelle 5.1 ist die Leistung des Ausgangssystems für diese Arbeit dargestellt. In der Beispielkonfiguration werden vier Echtzeitströme und Nicht-Echtzeitlast parallel betrieben. Mithilfe der Admission gemäß [HLR$^+$01] wurde der Zeitbedarf der Ströme ermittelt. In der Praxis ergeben sich sehr genaue Annäherungen an die gewünschte Qualität, nur der erste Strom wird leicht untererfüllt. In der Beispielkonfiguration bleibt aber *nicht viel* Bandbreite (1.119,52 kByte/s) für Nicht-Echtzeitaufträge übrig. Die Verwendung einer TCQ-Größe von eins spiegelt die Ausgangssituation dieser Arbeit wider.

In Tabelle 5.2 ist das Systemverhalten bei Benutzung von SSTF mit einer TCQ-Größe von zwei beschrieben. Hier ist gegenüber FIFO schon eine deutliche Leistungssteigerung zu erkennen. Alle Ströme werden leicht übererfüllt und es verbleibt eine Bandbreite von 4.959,67 kByte/s für Nicht-Echtzeitaufträge. Die gesamte gelieferte Bandbreite des Systems beträgt 9.932,15 kByte/s und ist damit etwa 1,5 mal so hoch wie bei FIFO. Das im vorherigen Abschnitt angesprochene Problem der Bestimmung der Verteilung für die Admission wurde — wie vorgeschlagen — iterativ mit drei Schritten gelöst.

Die größte Gesamtleistung wird bei der Benutzung von SATF erreicht. In Tabelle 5.3 sind die entsprechenden Werte angegeben. Auch hier wurde eine TCQ-Größe von zwei benutzt. Bei Verwendung von SATF werden in dieser Situation alle Ströme etwas übererfüllt. Trotzdem ist deutlich mehr Bandbreite (7.084,52 kByte/s) für Nicht-Echtzeitaufträge verfügbar als mit den vorher vorgestellten Verfahren. Die in dieser Konfiguration insgesamt zur Verfügung gestellte Bandbreite von 12.162,00 kByte/s ist etwas mehr als doppelt so groß wie bei der Verwendung von FIFO. Auch für diesen Versuch wurde die Verteilung für die Admission in drei Schritten iterativ bestimmt.

Strom	***Aufträge / Periode***	***gewünschte Qualität [%]***	***erreichte Qualität [%]***	***Bandbreite [kByte/s]***
1	20	95,00	94,82	2426,49
2	10	90,00	90,69	1160,46
3	5	85,00	86,59	554,00
4	10	60,00	62,44	799,02
Nicht-Echtzeit	100 (immer in der Warteschlange)	—	—	1119,52
gesamt	—	—	—	6059,51

Tabelle 5.1: FIFO, TCQ-Größe 1 (Platte: Seagate ST318406LW)

Strom	***Aufträge / Periode***	***gewünschte Qualität [%]***	***erreichte Qualität [%]***	***Bandbreite [kByte/s]***
1	20	95,00	95,30	2441,72
2	10	90,00	90,65	1161,29
3	5	85,00	86,59	554,70
4	10	60,00	63,60	814,76
Nicht-Echtzeit	100 (immer in der Warteschlange)	—	—	4959,67
gesamt	—	—	—	9932,15

Tabelle 5.2: SSTF, TCQ-Größe 2 (Platte: Seagate ST318406LW)

Strom	***Aufträge / Periode***	***gewünschte Qualität [%]***	***erreichte Qualität [%]***	***Bandbreite [kByte/s]***
1	20	95,00	95,94	2458,66
2	10	90,00	92,40	1183,85
3	5	85,00	88,30	565,65
4	10	60,00	67,84	869,30
Nicht-Echtzeit	100 (immer in der Warteschlange)	—	—	7084,52
gesamt	—	—	—	12162,00

Tabelle 5.3: SATF, TCQ-Größe 2 (Platte: Seagate ST318406LW)

5.4 Offene Fragen

Kurze Versuche mit ausschließlich sequentiellen Zugriffsmustern ergaben, dass der SATF-Algorithmus in der bisher beschriebenen Form eine deutlich schlechtere Leistung zeigt als FIFO. Der Grund für dieses Phänomen liegt an der etwas ungenauen Modellierung von sehr kurzen Seek-Bewegungen und vor allem an der Tatsache, dass Platten-Caches nicht im Modell berücksichtigt werden.

Durch eine Sonderbevorzugung von Plattenaufträgen, die auf die selbe Spur zielen, wie der aktuell abgeschlossene, konnte dieser Nachteil fast vollständig ausgeglichen werden. Dabei muss jedoch zwischen Schreib- und Leseaufträgen unterschieden werden, da der Schreib-Cache deaktiviert ist.

Die bessere, aber aufwendigere Lösung des Problems wäre die Erweiterung des mathematischen Plattenmodells um Platten-Caches, sodass eventuell noch mehr Vorteile daraus gezogen werden können.

Es gibt viele Faktoren, die Zugriffsmuster eines Gesamtsystems beeinflussen. Dazu gehören unter anderem das benutzte Dateisystem, die Größe und die Ersetzungsstrategie des Caches und die Anwendungen, die die Last erzeugen.

Die in dieser Arbeit benutzten zufälligen Plattenzugriffsmuster sind nicht gut dazu geeignet echtes Systemverhalten abzuschätzen, da sequentielle Zugriffe normalerweise mit einem nicht vernachlässigbaren Anteil auftreten. Sie sind aber sehr gut geeignet um die Unterschiede zwischen einzelnen Schedulingverfahren hervorzuheben.

Trotz diverser Optimierungen konnte der Prozessorzeitbedarf des Schedulers nicht auf null gesenkt werden. Insbesondere ist er natürlich auch von der Anzahl der in der Warteschlange befindlichen Aufträge abhängig. Es ist zu erwarten, dass sich — insbesondere auf langsamen Rechnern — bei sehr großen Warteschlangen negative Auswirkungen auf die Gesamtleistung der Platte ergeben, da die Overhead-Berechnung im mathematischen Modell von einer konstanten Rechenzeit im Scheduler ausgeht.

Für dieses Problem bieten sich mehrere Lösungsmöglichkeiten an:

- Das Problem kann ignoriert werden, da die Auswirkungen in der Praxis fast immer vernachlässigbar sind.
- Die Warteschlange, beziehungsweise die maximale Suchtiefe in ihr, kann beschränkt werden. Dadurch ergibt sich eine obere Zeitschranke für Schedulingentscheidungen.
- Das Modell könnte so erweitert werden, dass es seine eigene Rechenzeit berücksichtigt.

Bei der aktuellen Implementierung wird der erste Ansatz verfolgt. Für diese Vorgehensweise spricht auch die Tatsache, dass die Overhead-Parameter bei SATF deutlich weniger zum Tragen kommen wenn TCQ eingesetzt wird. Dieses sollte der Regelfall sein, da so die beste Leistung zu erzielen ist.

Trotz dieser Herangehensweise muss die Rechenzeit für den Plattenscheduler in einem Echtzeitsystem natürlich eingeplant werden.

5.5 Stand der Implementierung

Die Implementierung ist in der beschriebenen Form umgesetzt und läuft stabil. Des Weiteren wurde eine Reihe von Testprogrammen angepasst und erweitert, die u. a. für die Aufnahme der Graphen dieser Arbeit genutzt wurden. Der gewünschte Schedulingalgorithmus ist zur Laufzeit umschaltbar. Im Moment stehen FIFO, cSCAN, SSTF und SATF zur Verfügung. Die Implementierung weiterer Algorithmen dürfte problemlos möglich sein, da die entsprechende Aufrufschnittstelle einfach gehalten ist.

Bei Verwendung von SATF muss eine Geometriedatenbank mit Informationen über die Platte übergeben werden. Die Software zur Erstellung dieser Datenbank läuft weitestgehend automatisch. Lediglich für die Ermittlung der vier Overhead-Parameter ist ein Kalibrierungslauf unter DROPS notwendig. Der Erstellungsprozess ist aber aufgrund der umfangreichen Messungen relativ zeitaufwendig.

Die Ermittlung der notwendigen Informationen für die drei im Rahmen dieser Arbeit betrachteten Festplatten nahm jeweils etwa vier bis sechs Stunden in Anspruch.

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde eine Reihe verschiedener Plattenauftragsschedulingverfahren bezüglich ihrer Eignung für den DROPS SCSI-Treiber untersucht. Außerdem konnte ein Vorauswahlverfahren entwickelt werden, das es ermöglicht, verschiedene Schedulingalgorithmen in Echtzeitsystemen einzusetzen. Folglich wurde bei der Implementierung das Verfahren, das die höchste Leistungsfähigkeit verspricht (SATF), realisiert.

Der Vergleich mit der vorangegangenen Implementierung (FIFO) zeigt, dass eine deutliche Leistungssteigerung erreicht werden konnte. Sogar die für Vergleichszwecke implementierten weitverbreiteten Standardalgorithmen (cSCAN und SSTF, siehe auch Abschnitt 3.2) konnten noch deutlich übertroffen werden.

Vergleiche mit den in die Platte eingebauten Schedulingalgorithmen ergaben das überraschende Ergebnis, dass sich SATF bei zwei der drei untersuchten Platten mindestens ebenbürtig schlägt. Werden die notwendigen Randbedingungen für den Echtzeitbetrieb eingehalten (TCQ-Größe maximal 2) ist der Leistungsvorsprung bei allen drei Festplatten sehr deutlich.

Des Weiteren wurde die Extraktionssoftware zur Ermittlung der für SATF notwendigen Plateninformationen an SCSI angepasst. Sie läuft weitestgehend automatisch ab.

Für weitere Arbeiten bieten sich vielfältige Ansätze, von denen einige im Folgenden kurz beschrieben werden:

- Um das Verhungern von Nicht-Echtzeitaufträgen zu verhindern, kann entweder die Auftragsvorauswahl oder ein Schedulingalgorithmus selbst modifiziert werden. Das in Abschnitt 2.4.1 beschriebene Altern (*aging*) ist eine Ansatzmöglichkeit hierfür.
- Die Erstellung der Plattendatenbank mit den Geometrieinformationen, wie sie für den SATF-Algorithmus gebraucht werden, ist sehr zeitaufwendig. Außerdem lässt sie sich im Moment nur für eine Platte zugleich ermitteln und nicht für eine Plattenserie auf einmal. Hier bieten sich folgende Lösungsansätze an:
  - Man kann versuchen generelle Datenbanken für Plattenserien zu generieren und dann pro Platte nur die (hoffentlich kleinen) Differenzen ermitteln. Das ist vermutlich deutlich schneller als die Extraktion aller Merkmale, eventuell so schnell, dass es zur Laufzeit geschehen kann.

  - Bei der Datenbankerstellung kann auf Informationen aus den SCSI-Mode-Pages zurückgegriffen werden. U. u. reichen diese Informationen sogar für alle Aspekte der Datenbank aus, sodass auf empirische Messungen verzichtet werden kann. Wenn dadurch ein großer Geschwindigkeitsgewinn erzielt wird, kann die Ermittlung der kompletten Datenbank vielleicht zur Laufzeit erfolgen. Das würde eine große Einsatzhürde im praktischen Betrieb beseitigen.

- Die Speicherung der statischen Modellinformationen für den SATF-Algorithmus nimmt momentan relativ viel Platz im Hauptspeicher ein (je nach Platte etwa 1 MByte). Das ist für schnellen Zugriff auf die Geometrieinformationen notwendig. Hier könnte durch eine kompaktere Darstellung Speicher und vielleicht auch Rechenzeit eingespart werden.
  Die Seek-Funktion ist im Moment durch eine Tabelle mit einigen Stützpunkten implementiert. Eventuell bietet hier eine geschlossene Darstellung als Funktion Vorteile bezüglich Geschwindigkeit und Genauigkeit der Vorhersage.

- Das mathematische Modell, auf dem die Vorhersage der Bearbeitungszeit von Plattenaufträgen basiert, vereinfacht im Moment einige Eigenschaften von realen Platten. Eine Erweiterung des Modells könnte zu einer weiteren Leistungssteigerung führen. Folgende Festplatteneigenschaften kommen dafür in Frage:

  - *“Access-On-Arrival”*,
  - Platten-Caches (besonders der *“Read-Ahead-Cache”* bei sequentiellen Zugriffen),
  - mehrere Plattenköpfe (im Moment geht das Modell von einer Platte mit nur einem Schreib-Lese-Kopf aus) und
  - Auslagerung von fehlerhaften Sektoren (die Position der Ersatzsektoren ist im Moment nicht berücksichtigt).

- Für einige Anwendungen (zum Beispiel Journaling-Dateisysteme und Transaktionssysteme) ist die Möglichkeit von Barrieren im Auftragsstrom interessant. Das sind Aufträge, die nicht überholt werden dürfen und selbst nicht überholen. Solche Aufträge müssten in einem Scheduler natürlich besonders behandelt werden.

- Die Idee des OAT-Scheduling (siehe auch Abschnitt 3.2) muss nicht ganz verworfen werden. Eventuell könnte man für bestimmte Untermengen von Aufträgen berechnen, ob sie sicher zu schedulen sein werden. Für diese (kleineren) Untermengen lässt sich eventuell auch praktisch die optimale Bearbeitungsreihenfolge errechnen. Diese Berechnung kann auch optional in einem niedrig priorisierten Thread geschehen, sodass ansonsten nicht benötigte Rechenzeit dazu benutzt werden kann, die Plattenleistung weiter zu erhöhen.

Anhang A
Messwerte

In diesem Anhang werden die konkreten Messwerte von drei der untersuchten Platten dargestellt. Dabei ist die Firma Seagate mit zwei Festplatten und IBM mit einer Platte vertreten. Die Seagate-Platten haben eine Rotationsgeschwindigkeit von 7.200 beziehungsweise 10.000 u/min und die IBM-Platte dreht sich mit 15.000 u/min.

Es werden Graphen dargestellt, die die Leistung der verschiedenen Schedulingalgorithmen (FIFO, cSCAN, SSTF, SATF) unter verschiedenen Bedingungen (Größe der Warteschlange im Treiber, Größe der TCQ) vergleichen. Die Versuche, deren Messwerte hier dargestellt sind, wurden alle mit SIMPLE QUEUE TAGs (siehe auch Abschnitt 2.1.3) durchgeführt. Somit sind für den Echtzeitbetrieb nur TCQ-Größen bis zwei interessant, da die Platte einzelne Aufträge sonst umsortieren kann, wodurch theoretisch beliebig lange Verzögerungen ermöglicht werden.

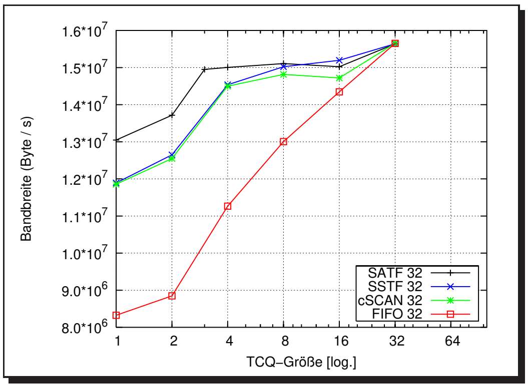
Die verwendeten Zufallszugriffsmuster sind über die ganze Platte gespreizt und bestehen aus einem ebenfalls zufälligen Gemisch aus Schreib- und Leseaufträgen. Die einzelnen Aufträge sind jeweils 64 kByte groß.

Die Darstellung der Ergebnisse des FIFO-Algorithmus dient dabei zwei Zielen:

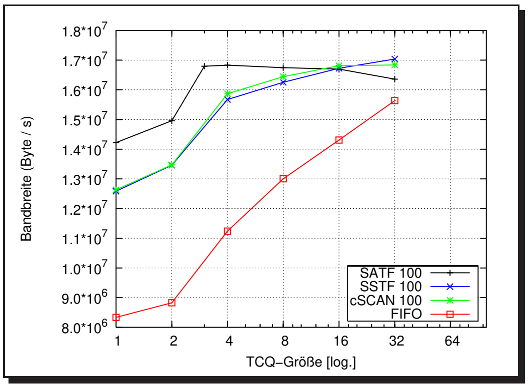
- Einerseits sind sie die Vergleichsbasis zur Ausgangssituation dieser Arbeit. Dafür ist nur der Messwert mit der TCQ-Größe eins relevant.
- Bei Verwendung höherer Werte für die TCQ-Größe kann der interne Plattenscheduler aktiv werden. Diese Werte sind also die Vergleichsgrundlage mit den hardwareinternen Algorithmen.

Alle untersuchten Platten unterstützen in Kombination mit dem verwendeten Kontroller und Treiber eine maximale TCQ-Größe von 32. Auch der platteneigene Speicher spielt bei der Begrenzung eine Rolle. Den internen Schedulingmöglichkeiten der Platten sind also schon auf dieser Ebene Grenzen gesetzt. Ein weiterer fundamentaler Unterschied zwischen dem platteninternen und dem externen Scheduling ist die Tatsache, dass einmal an die Platte abgeschickte Aufträge nicht mehr einfach abgebrochen werden können. Im Gegensatz dazu kann die Warteschlange des Treibers beliebig modifiziert werden. In Kombination mit den in Abschnitt 2.3.1 eingeführten normalen Echtzeitströmen mit Qualitätsparametern verbietet sich somit der Einsatz von TCQ, da nicht vorhergesagt werden kann, wie viele der in der Warteschlange befindlichen Aufträge tatsächlich auszuführen sind.

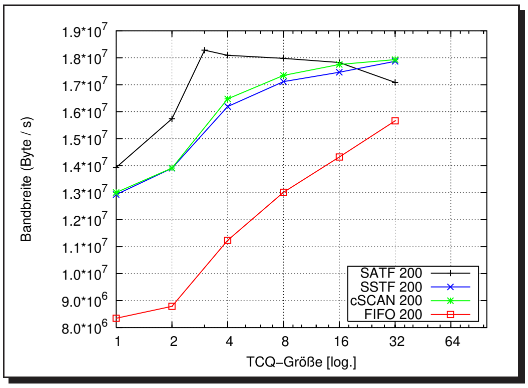
A.1 IC35L018UWPR15-0



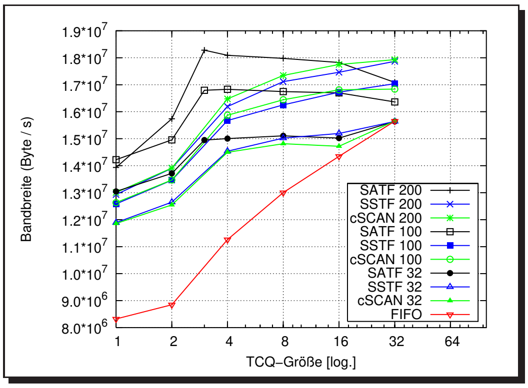
(a) 32 Aufträge in der Treiberwarteschlange



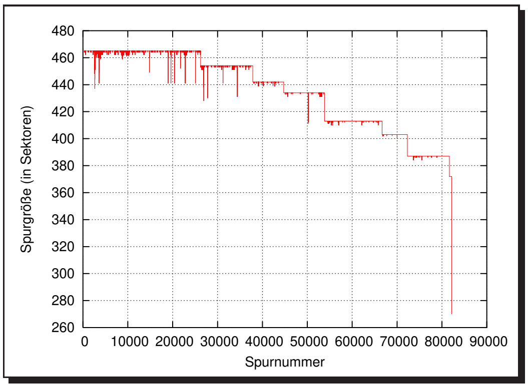
(b) 100 Aufträge in der Treiberwarteschlange



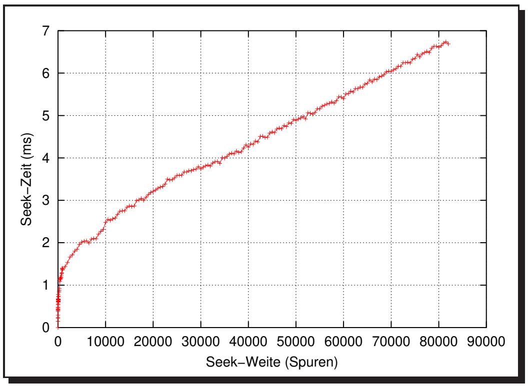
(c) 200 Aufträge in der Treiberwarteschlange



(d) 32, 100 bzw. 200 Aufträge in der Treiberwarteschlange



(e) Zoneneinteilung



(f) Seek-Kurve

Abbildung A.1: Leistungsübersicht für die Platte IC35L018UWPR15-0

Tabelle A.1: Herstellerangaben

Hersteller:	IBM
Modellbezeichnung:	Ultrastar 36Z15
Modellnummer:	IC35L018UWPR15-0
Seriennummer:	PTY1T984
Kapazität (LBA-Sektoren):	18,4 GB (35.843.670)
Cache-Größe:	4 MB
Sektorgröße:	512 Byte
Anschlusstyp:	Ultra160 SCSI
Rotationsgeschwindigkeit:	15.000 u/min
Köpfe:	8

Tabelle A.2: Gemessene und berechnete Werte

Kommando-Overhead:	0,77 ms
Korrekturwert für Lese-Lese-Aufträge (TCQ-Größe 1):	0,45 ms
Korrekturwert für Lese-Schreib-Aufträge (TCQ-Größe 1):	1,04 ms
Korrekturwert für Schreib-Lese-Aufträge (TCQ-Größe 1):	0,00 ms
Korrekturwert für Schreib-Schreib-Aufträge (TCQ-Größe 1):	0,40 ms
Korrekturwert für Lese-Lese-Aufträge (TCQ-Größe 2):	-0,05 ms
Korrekturwert für Lese-Schreib-Aufträge (TCQ-Größe 2):	0,55 ms
Korrekturwert für Schreib-Lese-Aufträge (TCQ-Größe 2):	0,20 ms
Korrekturwert für Schreib-Schreib-Aufträge (TCQ-Größe 2):	0,20 ms
Rotationszeit:	4,00 ms

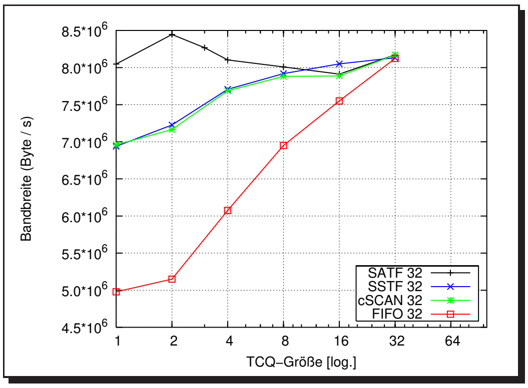
Die IBM-Platte war im Test die mit 15.000 u/min am schnellsten rotierende. Auch ihre gemessene maximale Seek-Zeit von ca. 7 ms ist minimal innerhalb der untersuchten Platten. Der interne Plattenscheduler dieser Platte hat sich bei den Versuchen als der beste erwiesen und übertrifft sogar SATF (siehe Abbildung A.1 ***(a)***, FIFO 32 bei TCQ-Größe 32 und SATF 32 bei TCQ-Größe 2).

Das hat vermutlich mehrere Gründe:

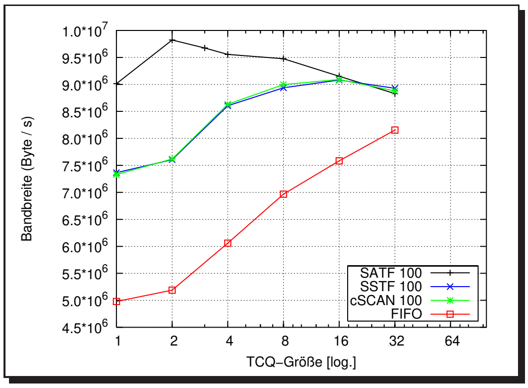
- Durch die extrem kurze Rotationszeit von nur 4 ms bleibt für den SATF-Algorithmus nicht viel Spielraum zum Optimieren.
- Die Platte hat mit acht Köpfen viermal so viele wie die beiden Seagate-Platten. Dadurch wird die Modellierung der Seek-Kurve ungenau, da Plattenköpfe im mathematischen Modell bisher nicht berücksichtigt werden. Die Ungenauigkeit tritt besonders bei kurzen Sprüngen zu Tage.

Interessant ist außerdem die Tatsache, dass zwischen TCQ-Größe zwei und drei noch eine starke Steigerung der gelieferten Plattenbandbreite zu beobachten ist (siehe Abbildung A.1 ***(a)***, ***(b)***, ***(c)*** und ***(d)***).

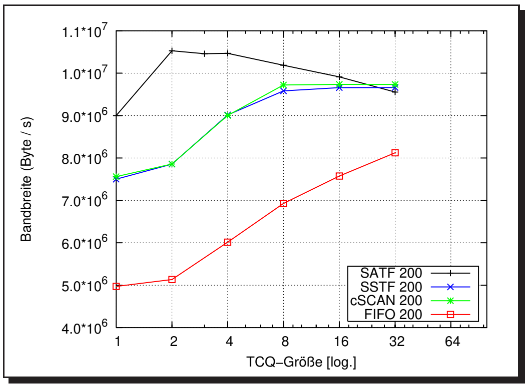
A.2 ST318438LW



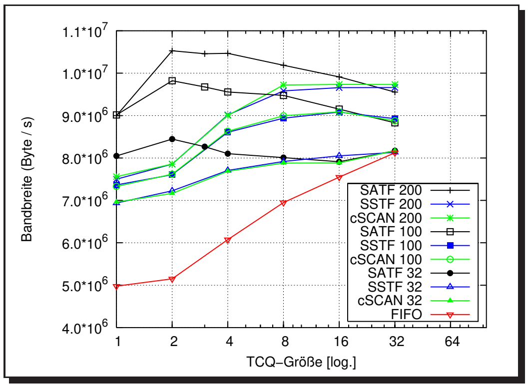
(a) 32 Aufträge in der Treiberwarteschlange



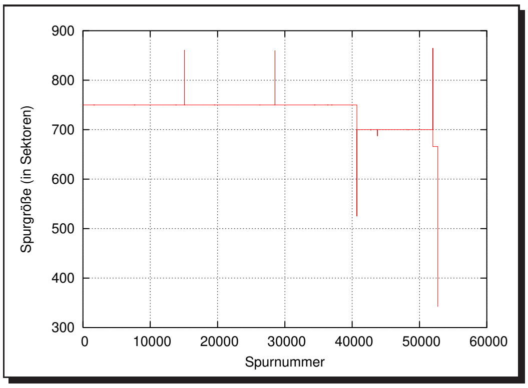
(b) 100 Aufträge in der Treiberwarteschlange



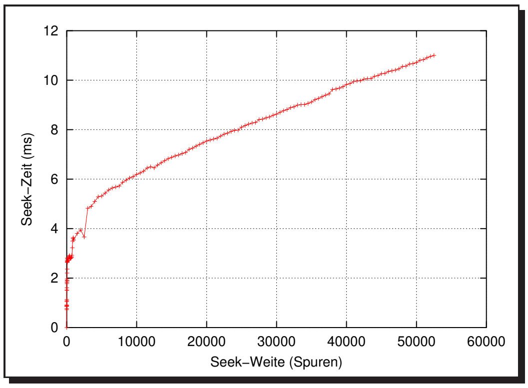
(c) 200 Aufträge in der Treiberwarteschlange



(d) 32, 100 bzw. 200 Aufträge in der Treiberwarteschlange



(e) Zoneneinteilung



(f) Seek-Kurve

Abbildung A.2: Leistungsübersicht für die Platte ST318438LW

Tabelle A.3: Herstellerangaben

Hersteller:	Seagate
Modellbezeichnung:	Barracuda 36ES2
Modellnummer:	ST318438LW
Seriennummer:	3JJ04G89000072495YWL
Kapazität (LBA-Sektoren):	19,924 GB (38.914.049)
Cache-Größe:	2 MB
Sektorgröße:	512 Byte
Anschlusstyp:	Ultra160 SCSI
Rotationsgeschwindigkeit:	7.200 u/min
Köpfe:	2

Tabelle A.4: Gemessene und berechnete Werte

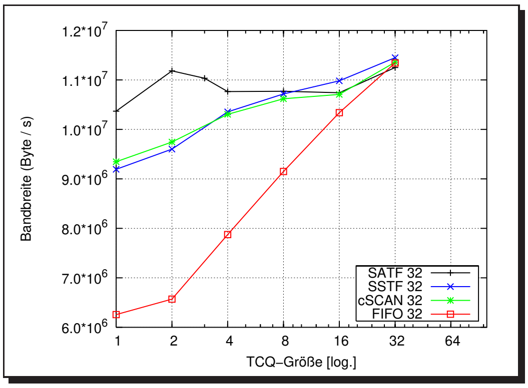
Kommando-Overhead:	0,56 ms
Korrekturwert für Lese-Lese-Aufträge (TCQ-Größe 1):	0,00 ms
Korrekturwert für Lese-Schreib-Aufträge (TCQ-Größe 1):	0,50 ms
Korrekturwert für Schreib-Lese-Aufträge (TCQ-Größe 1):	0,30 ms
Korrekturwert für Schreib-Schreib-Aufträge (TCQ-Größe 1):	0,30 ms
Korrekturwert für Lese-Lese-Aufträge (TCQ-Größe 2):	-0,25 ms
Korrekturwert für Lese-Schreib-Aufträge (TCQ-Größe 2):	0,00 ms
Korrekturwert für Schreib-Lese-Aufträge (TCQ-Größe 2):	-0,60 ms
Korrekturwert für Schreib-Schreib-Aufträge (TCQ-Größe 2):	-0,30 ms
Rotationszeit:	8,30 ms

Bei dieser relativ langsam drehenden Platte von Seagate hat der SATF-Scheduler gute Möglichkeiten, sich gegenüber cSCAN und SSTF abzusetzen. So ist der Leistungsabstand gegenüber den anderen Algorithmen schon bei einer Warteschlangenlänge von 32 deutlich (Abbildung A.2 ***(a)***). Bei Benutzung einer größeren Warteschlange (***(b)*** und ***(c)***) wird der Vorsprung noch deutlicher.

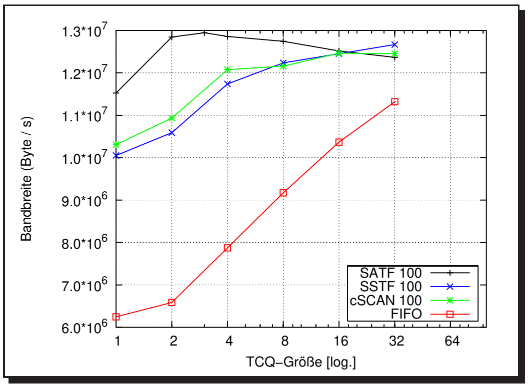
Ungewöhnlich ist die kleine Unregelmäßigkeit am Beginn der Seek-Kurve ***(f)***. Eventuell könnte hier der Leistungsvorsprung von SATF bei dieser Platte durch verbesserte Plattenparameter-Extraktionssoftware weiter ausgebaut werden.

Außerdem fällt auf, dass im Graph der Zoneneinteilung ***(e)*** unregelmäßige Ausschläge nach oben auftreten, das heißt einige Spuren größer als ihre Nachbarn ermittelt werden. Vermutlich werden von der Extraktionssoftware fälschlicherweise zwei kleinere Spuren zusammengefasst. Einzelne Spuren mit geringerer Sektorzahl als in ihrer Umgebung üblich deuten normalerweise auf ausgeblendete, fehlerhafte Oberflächenbereiche der Festplatte hin.

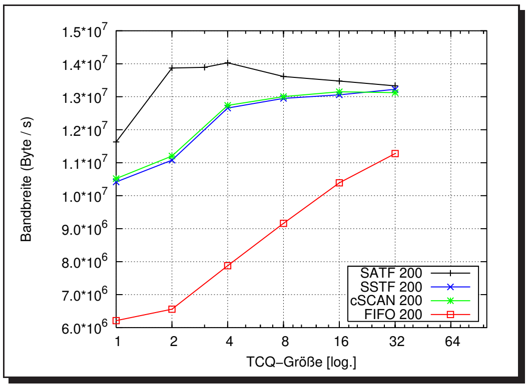
A.3 ST318406LW



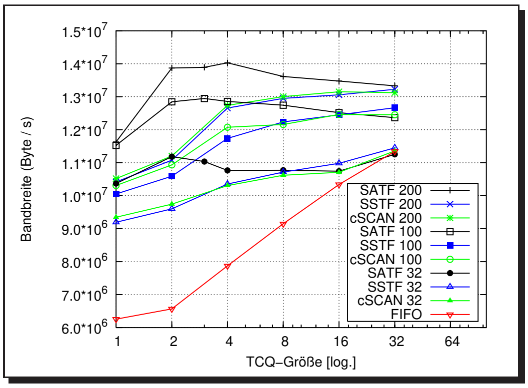
(a) 32 Aufträge in der Treiberwarteschlange



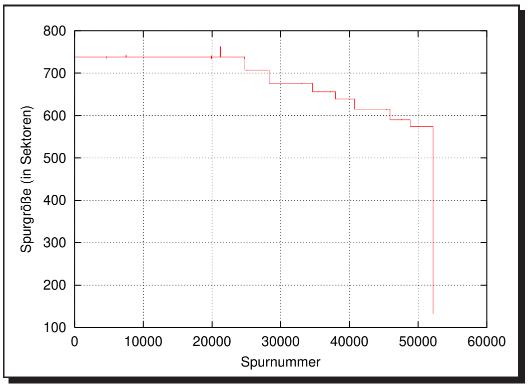
(b) 100 Aufträge in der Treiberwarteschlange



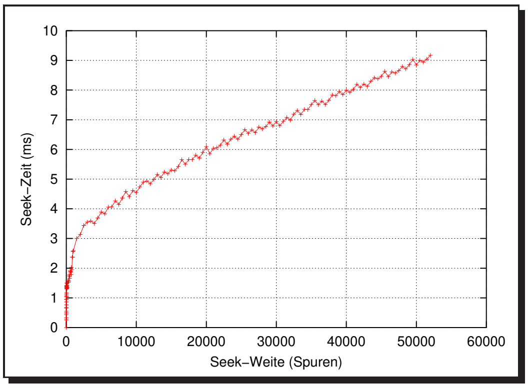
(c) 200 Aufträge in der Treiberwarteschlange



(d) 32, 100 bzw. 200 Aufträge in der Treiberwarteschlange



(e) Zoneneinteilung



(f) Seek-Kurve

Abbildung A.3: Leistungsübersicht für die Platte ST318406LW

Tabelle A.5: Herstellerangaben

Hersteller:	Seagate
Modellbezeichnung:	Cheetah
Modellnummer:	ST318406LW
Seriennummer:	3FE1QY6Q000073053GWX
Kapazität (LBA-Sektoren):	18,352 GB (35.843.670)
Cache-Größe:	4 MB
Sektorgröße:	512 Byte
Anschlusstyp:	Ultra160 SCSI
Rotationsgeschwindigkeit:	10.000 u/min
Köpfe:	2

Tabelle A.6: Gemessene und berechnete Werte

Kommando-Overhead:	0,61 ms
Korrekturwert für Lese-Lese-Aufträge (TCQ-Größe 1):	0,15 ms
Korrekturwert für Lese-Schreib-Aufträge (TCQ-Größe 1):	0,60 ms
Korrekturwert für Schreib-Lese-Aufträge (TCQ-Größe 1):	0,35 ms
Korrekturwert für Schreib-Schreib-Aufträge (TCQ-Größe 1):	0,60 ms
Korrekturwert für Lese-Lese-Aufträge (TCQ-Größe 2):	-0,20 ms
Korrekturwert für Lese-Schreib-Aufträge (TCQ-Größe 2):	0,10 ms
Korrekturwert für Schreib-Lese-Aufträge (TCQ-Größe 2):	-0,55 ms
Korrekturwert für Schreib-Schreib-Aufträge (TCQ-Größe 2):	-0,05 ms
Rotationszeit:	5,95 ms

Auch bei dieser schon recht schnell drehenden Platte liefert der SATF-Algorithmus in etwa die gleiche Leistung wie der interne Scheduler (siehe Abbildung A.3 ***(a)***) bei Verwendung gleicher Rahmenbedingungen (Warteschlangengröße auf 32 limitiert). Werden höhere Warteschlangengrößen verwendet (***(b)*** und ***(c)***), steigt der Vorsprung von SATF gegenüber allen anderen deutlich an.

Ansonsten fällt bei dieser Platte das leichte Rauschen der extrahierten Seek-Kurve ***(f)*** auf. Minimale Unregelmäßigkeiten nach oben in der Darstellung der Zoneneinteilung ***(e)*** deuten auch hier auf kleine Fehler in den von der Extraktionssoftware gelieferten Daten hin.

Anhang B

Glossar

Access-On-Arrival (auch *zero-latency-access*) Bei einem normalen Plattenzugriff wird der Schreib-Lese-Kopf auf den ersten Zielsektor positioniert. Danach erfolgt der Zugriff auf alle Zielsektoren des Auftrages sequentiell. Bei *“Access-On-Arrival”* kann die Platte die einzelnen Sektoren eines Auftrages in beliebiger Reihenfolge abarbeiten. Wenn zum Beispiel eine ganze Spur Ziel eines Auftrages ist, so kann der Zugriff direkt nach dem Seek erfolgen, somit entsteht keine Rotationsverzögerung.

Admission siehe Abschnitt 2.2.2 auf Seite 8

ATA ist die Abkürzung für “*Advanced Technology Attachment*”; eine Festplattenimplementierung, bei der der Kontroller in das Laufwerk integriert ist. Es existieren verschiedene Versionen von ATA, welche alle vom “*Small Form Factor (SFF) Committee*” entwickelt wurden. (vgl. [Jup03])

IDE ist die Abkürzung für “*Integrated Drive Electronics*”. An das IDE-Interface werden Massenspeichergeräte (zum Beispiel Festplatten und CD-ROM Laufwerke) angeschlossen, die einen integrierten Kontroller haben. (vgl. [Jup03])

lineares Mapping siehe Abschnitt 2.1.1 auf Seite 3

Scheduler siehe Abschnitt 2.2.1 auf Seite 7

SCSI ist die Abkürzung für “*small computer system interface*”. SCSI ist ein paralleler Interfacestandard, welcher in Apple Macintosh Rechnern, PCs und vielen UNIX-Systemen für den Anschluss von Peripheriegeräten (zum Beispiel auch Massenspeicher) genutzt wird. Im Rahmen dieser Arbeit wurden SCSI-Festplatten betrachtet. (vgl. [Jup03])

TCQ ist die Abkürzung für “*Tagged Command Queueing*” und wurde mit dem SCSI-2 Standard eingeführt, um es SCSI-Geräten zu ermöglichen mehrere Aufträge von einem Rechner gleichzeitig anzunehmen. Wenn die Aufträge im Pufferspeicher der Platte ankom-

men, werden sie mit einer Markierung (*Tag*) versehen, sodass der Prozessor der Festplatte umsortieren kann. Dieses Vorgehen kann die Kopfbewegungen der Platte verringern (vgl. Abschnitt 2.1.3). (vgl. [Jup03])
Für IDE-Festplatten existiert seit dem ATA-4 Standard mit dem *"queued feature set"* eine vergleichbare Lösung (vgl. [Tec97, Tec98, Tec02]). In der Praxis scheint sich deren Einsatz aber noch nicht durchgesetzt zu haben.

TCQ-Größe ist die Länge der Warteschlange in der Platte selbst. Die maximale TCQ-Größe einer Platte bestimmt, wie viele ausstehende Aufträge die Platte gleichzeitig puffern kann. Diese Größe wird vom Plattentreiber, vom SCSI-Kontroller und von der Platte selbst begrenzt. Neben der maximalen Größe der TCQ ist vor allem die tatsächliche Größe interessant. Sie kann auf verschiedene Arten begrenzt werden: Der Plattentreiber kann steuern, wie viele Aufträge er maximal gleichzeitig an die Platte gibt oder die Platte lehnt in einer bestimmten Situation neue Aufträge ab. Der zweite Fall kann auftreten, wenn der Pufferspeicher der Platte voll ist.

Worst-Case siehe Abschnitt 2.2.3 auf Seite 8

Literaturverzeichnis

[AAD98] Aboutabl, Mohamed, Ashok K. Agrawala und Jean-Dominique Decotignie: *Temporally Determinate Disk Access: An Experimental Approach (Extended Abstract)*. In: *Measurement and Modeling of Computer Systems*, Seiten 280–281, 1998.

[DT02] DROPS-Team: *DROPS - The Dresden Real-Time Operating System Project*. http://os.inf.tu-dresden.de/drops/, 2002.

[Gil] Gilbert, Douglas: *The Linux 2.4 SCSI subsystem HOWTO: raw devices*. http://www.linuxdoc.org/HOWTO/SCSI-2.4-HOWTO/rawdev.html.

[HC02] Huang, Lan und Tzi-Cker Chiueh: *Experiences in Building a Software-Based SATF Scheduler*. Technischer Bericht, July 2002.

[HLR+01] Hamann, C.-J., J. Löser, L. Reuther, S. Schönberg, J. Wolter und H. Härtig: *Quality Assuring Scheduling - Deploying Stochastic Behavior to Improve Resource Utilization*. In: *22nd IEEE Real-Time Systems Symposium (RTSS)*, London, UK, Dezember 2001.

[Jup03] Jupitermedia Corporation: *Webopedia: Online Dictionary for Computer and Internet Terms*. http://www.webopedia.com, 2003.

[JW91] Jacobson, David M. und John Wilkes: *Disk scheduling algorithms based on rotational position*. Technischer Bericht, HP Laboratories, 1991.

[Liu00] Liu, J. W. S.: *Real-Time Systems*. Prentice Hall, 2000.

[LSG02] Lumb, Christopher R., Jiri Schindler und Gregory R. Ganger: *Freeblock Scheduling Outside of Disk Firmware*. In: *Conference on File and Storage Technologies (FAST) January 28-30, 2002. Monterey, CA.*, Januar 2002.

[Meh98] Mehnert, Frank: *Ein zusagenfähiges SCSI-Subsystem für DROPS*. Diplomarbeit, TU Dresden, 1998.

[One75] ONEY, WALTER C.: *Queueing Analysis of the Scan Policy for Moving-Head Disks*. Journal of the ACM (JACM), 22(3):397–412, 1975.

[Poh02] POHLACK, MARTIN: *Ermittlung von Festplatten-Echtzeiteigenschaften*, Juni 2002. Großer Beleg.

[RW94] RUEMMLER, CHRIS und JOHN WILKES: *An introduction to disk drive modeling*. Technischer Bericht, Hewlett-Packard Laboratories, Palo Alto, CA, 1994.

[SCO90] SELTZER, M., P. CHEN und J. OUSTERHOUT: *Disk Scheduling Revisited*. In: *Proceedings of the USENIX Winter 1990 Technical Conference*, Seiten 313–324, Berkeley, CA, 1990. USENIX Association.

[Tec96] TECHNICAL COMMITTEE T10: *Information technology - Small Computer System Interface - 2, Working Draft*. Technischer Bericht, X3T9.2, 1996.

[Tec97] TECHNICAL COMMITTEE T13 AT ATTACHMENT: *Information Technology - AT Attachment-3 Interface (ATA-3) Revision 7b (DRAFT)*, 27 Januar 1997.

[Tec98] TECHNICAL COMMITTEE T13 AT ATTACHMENT: *Information Technology - AT Attachment with Packet Interface Extension (ATA/ATAPI-4) Revision 18 (DRAFT) (Revision 18)*, 19 August 1998.

[Tec02] TECHNICAL COMMITTEE T13 AT ATTACHMENT: *Information Technology - AT Attachment with Packet Interface - 6 (ATA/ATAPI-6) Revision 3b (DRAFT)*, 26 Februar 2002.

[WGP94] WORTHINGTON, B. L., G. R. GANGER und Y. N. PATT: *Scheduling Algorithms for Modern Disk Drives*. In: *Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Seiten 241–251, Nashville, TN, USA, 16–20 1994.

[Wil76] WILHELM, NEIL C.: *An anomaly in disk scheduling: a comparison of FCFS and SSTF seek scheduling using an empirical model for disk accesses*. Communications of the ACM, 19(1):13–17, 1976.