Verbesserung der Dateiverarbeitungskette in Betriebssystemen durch Nutzung der Skalierbarkeit moderner Kompressionsverfahren

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

Vorgelegt der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität Ilmenau

von
M. Comp. Sc. Heiko Sparenberg
geboren am 21. Juli 1977 in Mettingen

Gutachter:
1. Prof. Dr.-Ing. Karlheinz Brandenburg
   Technische Universität Ilmenau
2. Prof. Dr. Wolfram Schiffmann
   FernUniversität in Hagen
3. PD Dr.-Ing. Thomas Wittenberg
   Fraunhofer Institut für Integrierte Schaltungen (IIS)

Tag der Einreichung: 01.07.2014
Tag der wissenschaftlichen Aussprache: 23.04.2015

urn:nbn:de:gbv:ilm1-2015000152

Kurzfassung

Motiviert durch die aktuellen Herausforderungen im Bereich der computergestützten Bearbeitung vom Multimediadaten, leistet die vorliegende Arbeit einen Beitrag zum Forschungsgebiet der Datenverarbeitung und Dateiverwaltung innerhalb von Computersystemen durch neuartige Verfahren zur Nutzung skalierbarer Medien unter Verwendung vorhandener Datei- und Betriebssysteme.

Hierzu werden die Kompressionsformate JPEG 2000 und H.264 SVC vorgestellt und gezeigt, wie die Eigenschaft der Skalierbarkeit innerhalb der verschiedenen Verfahren erreicht wird. Es folgt eine Analyse der limitierenden Hard- und Softwarekomponenten in einem Computersystem für das o. g. Einsatzgebiet, bei der die Einschränkungen genutzter Datenspeicher, Schnittstellen und Dateisysteme, sowie die heute üblichen Techniken zur Umgehung dieser Limitierungen, dargestellt werden. Ausgehend vom hohen Aufwand zur Kompensation der Leistungsengpässe werden anschließend neue Lösungsansätze unter Nutzung skalierbarer Medienformate abgeleitet, die nachfolgend untersucht werden.

Die vorliegende Arbeit zeigt hierzu neue Konzepte zur Verwaltung skalierbarer Mediendaten, die durch ein neues Rechtemanagement sowie durch eine speicheradaptive Ablagestrategie abgedeckt werden. Das Rechtemanagement erlaubt die Vergabe von Zugriffsrechten auf verschiedene Abschnitte einer Datei, wodurch die Skalierbarkeit der Medien derart abgebildet werden kann, dass verschiedene Benutzer — in Abhängigkeit ihrer Zugriffsrechte — unterschiedliche Varianten einer Datei angezeigt bekommen. Die speicheradaptive Ablagestrategie erreicht Durchsatzsteigerungen der verwendeten Datenträger, wenn das spätere Zugriffsverhalten auf die gespeicherten Medien vorab bekannt ist.

Weiter werden Verbesserungen der Verarbeitungsabläufe unter Ausnutzung skalierbarer Medien gezeigt. Auf Basis der entwickelten Substitutionsmethode zur Kompensation fehlender Daten einer skalierbaren Datei wird eine echtzeitfähige Einlesestrategie vorgestellt, die unzureichende Durchsatzraten von Speichermedien bzw. langsamen Schnittstellen derart kompensieren kann, dass eine unterbrechungsfreie Ausspielung von Bildsequenzen bei einer vorgegebenen Bildwiederholrate gewährleistet werden kann. Angepasste Cache-Strategien ermöglichen eine Steigerung der im Cache vorhaltbaren Einzelbilder im Vergleich zu nicht skalierbaren Varianten. Darüber hinaus wird das Konzept eines parametrisierbaren Dateiaufrufes eingeführt, wodurch mittels Zusatzinformationen im virtuellen Dateinamen eine gewünschte Variante einer skalierbaren Datei vom Datenspeicher angefragt werden kann.

Ein Vergleich mit dem Stand der Technik und eine Diskussion der Ergebnisse sowie einer kritischen Beurteilung der entwickelten Verfahren geben einen Ausblick für zukünftige wissenschaftliche Arbeiten auf dem adressierten Forschungsgebiet.

Abstract

Motivated by the current challenges in the field of computerized processing of multimedia information, this work contributes to the field of research on data processing and file management within computer systems. It presents novel techniques that enhance existing file- and operating systems by utilizing the scalability of modern media formats.

For this purpose, the compression formats JPEG 2000 and H.264 SVC will be presented with a focus on how they achieve scalability. An analysis of the limiting hard- and software components in a computer system for the application area is presented. In particular, the restrictions of the utilized storage-devices, data interfaces and file systems are laid out and workarounds to compensate the performance bottlenecks are depicted. According to the observation that compensating the defiles requires extra efforts, new solution statements utilizing scalable media are derived and examined, subsequently.

The present work reveals new concepts for managing scalable media files comprising a new rights management as well as a use-case-optimized storage technique. The rights management allows for granting access to certain parts of a file by which the scalability of the media files can be exploited in a way that users get access to various variants depending on their access rights. The use-case-optimized storage technique increases the throughput of hard disk drives when the subsequent access pattern to the media content is known a-priori.

In addition, enhancements for existing data workflows are proposed by taking advantage of scalable media. Based on the *Substitution Strategy*, where missing data from a scalable file is compensated, a real-time capable procedure for reading files is shown. Using this strategy, image-sequences comprising a video can be shown at a given frame rate without interruptions caused by insufficient throughput of the storage device or low-speed interfaces used to connect the storage device. Adapted caching-strategies facilitate an increase of images residing in cache compared to non-scalable-variants. Additionally, a concept called *Parameterizable File-Access* is introduced which allows users to request a certain variant of a scalable file directly from the file system by adding side-information to a virtual file name.

A comparison with the state of the art, a discussion of the results and a critical assessment of the methods developed in this work provide an outlook for future scientific work on the addressed research area.

Danksagung

Ich möchte mich an dieser Stelle bei allen bedanken, die direkt oder indirekt zur Entstehung dieser Arbeit beigetragen haben. Meinem Doktorvater Prof. Dr.-Ing. Dr. rer. nat. h.c. mult. Karlheinz Brandenburg für die Möglichkeit der Durchführung der Arbeit im Fachgebiet Elektronische Medientechnik.

Weiter danke ich Herrn Prof. Dr. Wolfram Schiffmann und PD Dr.-Ing. Thomas Wittenberg für ihr Interesse an meiner Arbeit, das sie durch die Übernahme des Zweit- bzw. Drittgutachtens zum Ausdruck gebracht haben.

Ich danke meinen Kollegen am Fraunhofer IIS für ihre Motivation und Unterstützung, vor allem Alexander Schmitt, dessen Ideen entscheidend zur Motivation dieser Arbeit beigetragen haben sowie Dr. Siegfried Fößel für die Schaffung der nötigen Rahmenbedingungen im Institut. Ich danke meinen Diplomanden bzw. Bachelor- und Masterstudenten Robert Scheler, Matthias Martin, Simon Huber, Tobias Joormann und Simon Beer für ihre wertvollen Denk- und Programmierarbeiten. Meinen Kollegen Volker Bruns, Carsten Feldheim, Sebastian Marshall und Matthias Martin danke ich für die anregenden Diskussionen bei der Erstellung dieser Arbeit.

Mein größter Dank gilt meiner Frau Ina für die großartige Unterstützung in den letzten Jahren.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Bei der Auswahl und Auswertung folgenden Materials haben mir die nachstehend aufgeführten Personen in der jeweils beschriebenen Weise unentgeltlich geholfen:

1. Herr Dipl.-Inf. Matthias Martin (Erfassung der Durchsatzraten der in Abschnitt 5.3 und 5.4 vorgestellten Algorithmen für die Cache-Verfahren für skalierbare Mediendaten).

Weitere Personen waren an der inhaltlich-materiellen Erstellung der vorliegenden Arbeit nicht beteiligt. Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- bzw. Beratungsdiensten (Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalte der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer Prüfungsbehörde vorgelegt.

Ich bin darauf hingewiesen worden, dass die Unrichtigkeit der vorstehenden Erklärung als Täuschungsversuch angesehen wird und den erfolglosen Abbruch des Promotionsverfahrens zu Folge hat.

_____ _____

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

1. Einleitung

Immer mehr Kompressionsverfahren für digital gespeicherte Bilder, Musikstücke und Videodaten — nachfolgend Multimediadaten genannt — verfügen über die Eigenschaft der Skalierbarkeit. Grundidee dieser Skalierbarkeit ist es, einen komprimierten Datenstrom zu generieren, aus dem sich anschließend sog. *Subvarianten* extrahieren lassen, die jeweils eine reduzierte Repräsentation der Mediendaten darstellen, ohne den encodierten Datenstrom dafür in seiner Gesamtheit decodieren zu müssen. Für skalierbar komprimierte Einzelbilder sind u. a. Subvarianten denkbar, die eine reduzierte Bildgröße und/oder Bildqualität beinhalten. Bereits bei der Encodierung wird festgelegt, welche Subvarianten später im komprimierten Datenstrom verfügbar sind. Der Decoder erkennt, welche Datenbereiche für eine gewünschte Subvariante decodiert werden müssen. Nicht benötigte Bereiche werden vom Decoder übersprungen, was üblicherweise zu einer schnelleren Decodierung und somit zur schnelleren Erstellung einer Subvariante führt. Demgegenüber steht die Gruppe der nicht-skalierbaren Kompressionsverfahren, die keine Erstellung von Subvarianten zulässt. Hier ist lediglich die Dekomprimierung des kompletten Datenstroms möglich, um die einzig verfügbare Variante zu erhalten.

Skalierbare Kompressionsverfahren bieten sich an, um Leistungsengpässe in der Verarbeitungskette von Multimediadaten zu kompensieren. Bei der Übertragung über ein Datennetzwerk kann bspw. eine reduzierte Subvariante übertragen werden, wenn die Bandbreite nicht für alle komprimierten Daten ausreicht und die Übertragung zeitkritisch ist. Bei der Decodierung eines Videos auf einem beliebigen Endgerät kann ebenfalls eine Subvariante gewählt werden, wenn die verfügbaren Rechenressourcen nicht ausreichen, um das Video in voller Auflösung/Qualität und ohne Bildaussetzer abzuspielen. Auch wenn die Effizienz skalierbarer Kompressionsverfahren für die oben genannten Multimediadaten in den letzten Jahren sukzessiv verbessert wurde und heute vergleichbar mit den nicht-skalierbaren Varianten ist, kann bisher lediglich eine geringe Verbreitung der skalierbaren Codecs festgestellt werden.

Bereits das 1992 vorgestellte Kompressionsformat der *Joint Photographic Expert Group* (JPEG), entsprechend bekannt als das JPEG-Format [7,82,125], bietet mit der optionalen Erweiterung *Progressive JPEG* eine Qualitätsskalierbarkeit, die es ermöglicht, ein Bild bereits vor Fertigstellung der Übertragung mit reduzierter Qualität anzuzeigen und durch nachgelieferte Daten zu verfeinern. Auch wenn die Dateigröße nicht signifikant größer wird [110], hat sich Progressive-JPEG nicht durchsetzen können, weil heutige Datennetze üblicherweise keinen Engpass für Einzelbilder darstellen und die Generierung von kleinen Vorschaubildern (sog. *Thumbs*) eine gängige Alternative bietet. Beim Nachfolgeformat

JPEG 2000 [41] wurde großes Augenmerk auf die Skalierbarkeit gelegt, um die o. g. Herausforderungen bei der Netzwerkübertragung adressieren zu können. Auch wenn die Unterstützung in gängiger Fotobearbeitungssoftware mittlerweile sehr umfangreich ist und JPEG 2000 sowohl im Bereich der Medizintechnik [73], bei der Distribution digitaler Kinofilme [18] und der digitalen Filmarchivierung [42] angewendet wird, ist die Verbreitung von JPEG 2000 zum heutigen Zeitpunkt im Vergleich zum Standard-JPEG noch gering. Hauptgrund hierfür liegt vor allem beim hohen Rechenaufwand zur Erstellung und Decodierung der Bilder. Zur Beschleunigung werden üblicherweise, sowohl für die Encodierung wie auch für die Decodierung, Mehrkernprozessoren, dedizierte Beschleunigerkarten [2,74,93] oder Grafikkarten [5] verwendet. Diese Möglichkeiten sind allerdings auf mobilen Geräten, wie z. B. Digitalkameras oder Smartphones, kaum vorhanden. Auch im Bereich der Videocodierung wurden bereits in den Coding-Standards *H.262/MPEG-2* [47], *H.263* [48] sowie *MPEG-4 Visual* [39] verschiedene Skalierungsoptionen eingeführt, die es erlauben, Subvideos mit reduzierter zeitlicher und räumlicher Auflösung oder Qualität zu extrahieren. Jedoch wurden die Videoströme durch die zusätzlichen Optionen größer und die benötigte Algorithmik im Decoder komplizierter als in der Standardvariante ohne Skalierung, weshalb keine nennenswerte Verbreitung erreicht werden konnte. Das aktuelle Verfahren *H.264 Scalable Video Coding (SVC)* [97,98] behebt die Nachteile seiner Vorgänger und findet erste Anwendungen im Bereich Live-Streaming [36,116]. Video-on-Demand-Systeme favorisieren hingegen adaptive Streaming-Verfahren, bei denen ein Video in mehreren Varianten vorliegt und ein Klient, in Abhängigkeit seiner aktuellen Netzwerkanbindung oder Darstellungsmöglichkeiten, nahezu beliebig zwischen den Varianten umschalten kann.

Ähnlich verhält es sich bei der Audio-Codierung, bei der das Konzept der Skalierbarkeit im Zeitbereich in [3] vorgestellt und daraus abgeleitete Erweiterungen zur Qualitätsskalierung im Frequenzbereich [20,26] in *MPEG-4 AAC Scalable* [24,40] übernommen wurden. Weder die aktuellste Variante *MPEG-4 Scalable to Lossless (SLS)* [132] noch das proprietäre *OptimFROG* [21] sowie das freie Format *WavPack* [6] konnten bisher weitreichende Akzeptanz erreichen. Wie bei Progressive JPEG sind die Gründe auch hier in der Verfügbarkeit breitbandiger Netzwerke zu finden.

Auch wenn die Anwendung der Skalierbarkeit bisher noch keine breite Durchdringung erzielen konnte, ist sie seit vielen Jahren intensives Forschungsgebiet, besonders im Bereich der Übertragung von Multimediadaten über Datennetze. Ebenfalls findet die Skalierbarkeit Anwendung in existierender Hard- und Software, die eine explizite Kenntnis der Skalierbarkeit aufweist und verschiedene Varianten extrahieren kann. Somit gibt es aktuell Verfahren zur Anlieferung der Daten über Datennetzwerke sowie diverse Anwendungen auf Applikationsebene, die sich die Skalierbarkeit zu Nutze machen. Bei der Verarbeitung der Daten innerhalb der Computerarchitektur — vor allem bei der Speicherung, Verwaltung und beim Datentransfer zwischen den einzelnen Komponenten — wird bisher nur unzureichend Gebrauch von der Skalierbarkeit gemacht. Dabei liefert die Technologie gute Voraussetzun-

gen, um auch innerhalb der Computerarchitektur auftretende Engpässe zu umgehen. Allerdings unterscheiden sich die Rahmenbedingungen im Vergleich zur Übertragung von Dateien über Datennetzwerke, weshalb eine simple Adaption der vorhandenen Technologien aus dem Web-Umfeld nicht möglich ist.

Die vorliegende Arbeit befasst sich demnach nicht mit der Übertragung skalierbarer Multimediadaten über Datennetzwerke, sondern adressiert den bisher unzureichend betrachteten Bereich der Nutzung skalierbarer Medien innerhalb einer Computerarchitektur. Im Besonderen werden dabei heutige Herausforderungen in der professionellen und semiprofessionellen Filmproduktion, sowie der Langzeitarchivierung von digitalen Filmdaten betrachtet. Charakteristisch für das o. g. Einsatzgebiet ist die computergestützte Verarbeitung von Multimediadaten, die üblicherweise in hoher Qualität und hoher räumlicher Auflösung vorliegen. Die Hardware-Komponenten der verwendeten Computer können dabei schnell an ihre Leistungsgrenzen stoßen, besonders wenn ein Abspielen ohne Bildaussetzer in verschiedenen Arbeitsschritten gewünscht ist. Typische Datenraten liegen zwischen 30 MBit/s und 1 GBit/s, wobei eine Videosequenz häufig in verschiedenen Auflösungen und Qualitäten bereitgestellt wird, um dadurch den Anforderungen verschiedener Benutzergruppen gerecht zu werden.

Für eine effektive Bearbeitungsmöglichkeit dieser Daten werden Videosequenzen üblicherweise als Einzelbilder durch die sog. *Intra-Frame-Codierung* abgelegt. Hierbei ist jedes einzelne Bild vollständig beschrieben und kann eigenständig, ohne Auswertung der Nachbarbilder in der Videosequenz, decodiert und angezeigt werden. Hierdurch ergeben sich entsprechende Vorteile für die computergestützte Bearbeitung der Filmsequenz, wie bspw. ein wahlfreies Anwählen von Bildern für den Prozess des Filmschnitts. Dem gegenüber steht die sog. *Inter-Frame-Codierung.* Hier existieren neben vollständig beschriebenen Bildern (Intra-Bild bzw. *I-Frame*), noch der Typ der Differenzbilder (Inter-Bild), die lediglich Änderungen zu vorhergehenden sowie nachfolgenden Bildern beinhalten[1]. Da aufeinanderfolgende Bilder in einer Bildsequenz üblicherweise eine hohe Ähnlichkeit aufweisen, kann durch eine Inter-Frame-Codierung eine höhere Kompressionseffizienz im Vergleich zur Intra-Frame-Codierung erreicht werden – ein bildgenaues Anwählen ist allerdings nur auf Intra-Bildern möglich. Der Fokus der vorliegenden Arbeit liegt deshalb im Bereich der Intra-Frame-Codierung.

Zur Vereinfachung der Verwaltung werden die komprimierten Einzelbilder der Filmsequenz üblicherweise in einer Containerdatei zusammengefasst. Im o. g. Einsatzgebiet wird dazu vorrangig das sog. *Material eXchange Format* (MXF) verwendet. Dies wirkt sich nicht auf

[1] Sog. *P-Frames* speichern Unterschiede zu vorhergehenden Bildern. *B-Frames* speichern Unterschiede zu vorhergehenden und nachfolgenden Bildern.

Ausgehend von den o. g. Einsatzgebieten und den gezeigten Rahmenbedingungen werden zwei Anwendungsschwerpunkte adressiert:

1. Neue Konzepte zur Verwaltung skalierbarer Medien durch Dateisysteme, die per Design Kenntnis über die Skalierbarkeit verwalteter Multimediadaten besitzen.

2. Echtzeitfähige Verfahren für Einzelplatzrechner, bei denen Engpässe vorhandener Hard- und Softwarekomponenten, wie z. B. langsame Datenträger, bandbreitenlimitierte Schnittstellen oder geringe Prozessorleistungen, unter Nutzung der Skalierbarkeit kompensiert werden. Echtzeitfähigkeit beschreibt in diesem Kontext, dass ein computergestütztes Abspielsystem in der Lage ist, ein Video — bzw. eine Sequenz von Einzelbildern — ohne Bildaussetzer mit einer bestimmten Bildwiederholrate wiederzugeben. Eine Verzögerung (Latenz), bspw. zwischen dem Start der Ausspielung und der eigentlichen Darstellung auf einem Bildschirm, ist dabei akzeptabel, sollte aber im Bereich weniger Sekunden liegen, um eine benutzerfreundliche Anwendbarkeit der Verfahren zu ermöglichen.

Die dazu vorgestellten Verfahren zeigen ein neues Anwendungsgebiet für skalierbare Mediendaten, können somit zur weiteren Verbreitung der verfügbaren Technik beitragen und dem gesamten Themengebiet dadurch hoffentlich einen positiven Impuls verleihen.

Basierend auf diesen Algorithmen lassen sich verschiedene Betriebs- und Dateisysteme entwickeln bzw. vorhandene Systeme erweitern, die dadurch per Design Kenntnis über skalierbare Medienformate besitzen und Daten effizienter speichern, abrufen und verwalten können. Darüber hinaus können neue Rechtesysteme für skalierbare Datenformate entwickelt und in bestehende Systeme integriert werden.

2. Grundlagen, Einflussfaktoren und Motivation für diese Arbeit

Dieses Kapitel zeigt zunächst die Funktionsweise der für diese Arbeit wichtigen, skalierbaren Medienformate und erläutert, wie Skalierung bei der Kompression und Dekompression erreicht wird. Anschließend werden vorhandene Speicher- und Dateisysteme, Schnittstellen und Cache-Verfahren innerhalb eines Computersystems (vgl. Abbildung 1) vorgestellt und deren Einflussfaktoren auf die Gesamtleistung erläutert. Entscheidend dabei ist, dass verschiedene Einflussfaktoren die gewünschte Echtzeitausspielung von Multimediadaten verhindern. Entsprechend sind die Herausforderungen sowie die auftretenden Problemstellungen bei der Verwendung verfügbarer Technologien in der Filmproduktion und Filmdistribution im dritten Abschnitt dieses Kapitels zusammengefasst. Hieraus ergibt sich die Motivation für die Entwicklung neuer Verfahren, die eine Skalierbarkeit der verwendeten Mediendaten innerhalb heutiger Computer-Architekturen zur Kompensation der Problemstellungen nutzen. Diese Verfahren werden in den nachfolgenden Kapiteln vorgestellt.

Der sonst übliche Teil zum Stand der Technik ist an dieser Stelle nicht zusammenfassend aufgeführt, sondern wird in den jeweiligen Kapiteln separat erläutert.

2.1 Wichtige Kompressionsformate

Skalierbare bzw. hierarchische Bildkompressionsformate bieten die Möglichkeit, verschiedene Varianten einer Datei aus der originalen Datei extrahieren zu können, ohne dafür den encodierten Datenstrom in seiner Gesamtheit decodieren zu müssen. Diese Variante kann verschiedene Eigenschaften besitzen, die bereits bei der Kompression festgelegt werden. Für Einzelbildformate sind Skalierungen der Bildgröße, Bildqualität, Anzahl der Farbkomponenten oder die Variation der Datenrate für bestimmte Bildregionen (engl. *Region Of Interest – ROI*) üblich. Für Bildsequenzen bzw. Videos sind verschiedene Varianten für Bildgröße, Bildqualität oder Bildwiederholrate üblich. Für Audiodaten sind verschiedene Audioqualitäten relevant.

Die Skalierbarkeit kann dabei in vielen Bereichen der digitalen Datenverarbeitung genutzt werden. Unzureichende Rechenleistung bei der Wiedergabe kann bspw. dadurch kompensiert werden, dass eine Variante mit geringerer Auflösung oder Qualität decodiert wird. Bei Videos lässt sich zusätzlich noch die Bildwiederholrate verringern. Engpässe in Datennetzen lassen sich ebenfalls durch Skalierung kompensieren, indem bei abnehmender Verfügbarkeit der Datenrate eine Variante mit geringerer Auflösung oder Qualität übertragen wird, was direkten

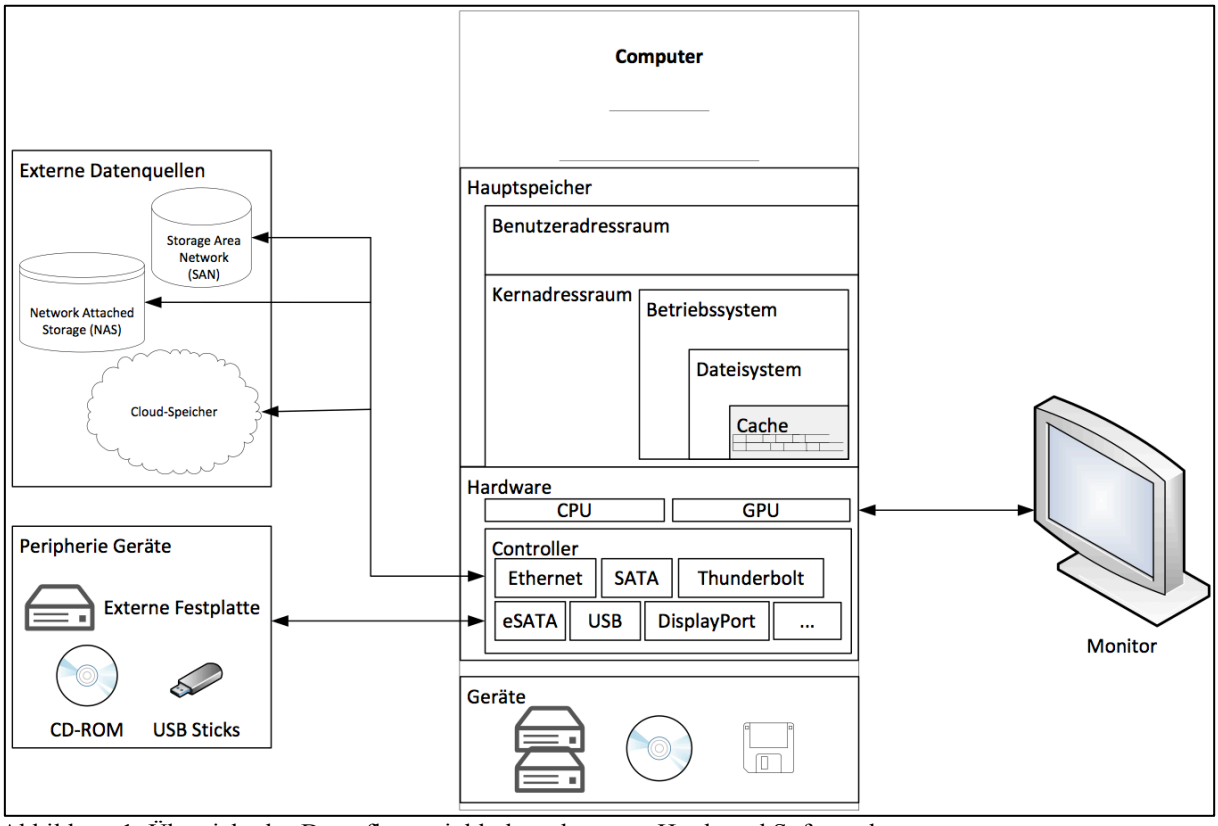

Abbildung 1: Übersicht des Datenflusses inkl. der relevanten Hard- und Softwarekomponenten

Einfluss auf die zu übertragene Datenmenge hat. Ohne skalierbare Verfahren müssen verschiedene Varianten einer Datei — sog. *Proxies* — vorab erstellt werden. Neben dem höheren Aufwand durch die Mehrfachkomprimierung und dem zusätzlichen Bedarf an Speicherressourcen, steigt der Aufwand zur Verwaltung dieser Proxy-Dateien.

Nachfolgend werden die Kompressionsverfahren JPEG und dessen Nachfolger JPEG 2000, die Familie der H.26x- bzw. MPEG-x-Standards — mit besonderem Fokus auf den aktuellen, skalierbaren Codec H.264 SVC — sowie eine Übersicht skalierbarer Audio-Codecs aufgezeigt.

2.1.1 JPEG

Eine erste Variante der Skalierbarkeit für komprimierte Einzelbilder wurde 1992 mit dem JPEG-Standard [7,82,125] eingeführt. Die optionale Erweiterung *Progressive JPEG* liefert eine besondere Form der Qualitätsskalierung. Dadurch kann ein Bild bereits in voller Größe und verminderter Qualität angezeigt werden, auch wenn noch nicht sämtliche Daten der Datei vorliegen. Treffen anschließend Daten für weitere Qualitätsschichten ein, wird das Bild erneut decodiert und dadurch die Qualität des Bildes verfeinert. Die benötigte Datenmenge ist bei Progressive JPEG nicht zwangsläufig größer als bei Standard bzw. *Baseline-JPEG*-Bildern. In vielen Fällen werden die Bilder sogar kleiner [110]. Hauptziel von Progressive-JPEG war der

Einsatz auf Webseiten, wodurch die Nutzer eingebettete Bilder entsprechend schneller angezeigt bekommen. Durch die Tatsache, dass die Bilder nach Erhalt weiterer Daten erneut komplett decodiert werden müssen [125], stellt Progressive JPEG keine echte Variante der Skalierbarkeit dar.

2.1.2 JPEG 2000

Im Jahr 1997 wurde von der JPEG ein sog. *Call for Proposals* [45] veröffentlicht, der Vorschläge zur Verbesserung der Probleme damals aktueller Kompressionsverfahren für Einzelbilder liefern sollte. Aus dieser Initiative entwickelte sich das Kompressionsformat JPEG 2000, das anschließend durch die *International Standardization Organization (ISO)* standardisiert wurde [41,42]. Dieser Standard beinhaltet aktuell 14 verschiedene Elemente, von denen hier lediglich Teil 1 (engl. *Part 1*) [41] betrachtet werden soll. Abbildung 2 zeigt die Verarbeitungsschritte während der JPEG 2000-Encodierung.

Vorverarbeitung:

Optional kann das unkomprimierte Bild in beliebig viele, nicht überlappende Kacheln (sog. *Tiles*) aufgeteilt werden. Die Größe der einzelnen Kacheln kann variieren, ebenso kann das komplette Bild in einer einzigen Kachel untergebracht sein. Jede Kachel wird anschließend separat — bei Bedarf auch mit unterschiedlichen Kompressionsparametern — weiterverarbeitet. Im Folgenden wird davon ausgegangen, dass das optionale Aufteilen in Kacheln keine Verwendung findet. Zur Vereinfachung der Implementierung nachfolgender Verarbeitungsschritte werden die Eingangsdaten symmetrisch um den Wert 0 verteilt. Für nichtvorzeichenbehaftete Eingangswerte wird hierzu ein sog. *DC-Offset* durchgeführt, wobei von jedem Eingangswert des unkomprimierten Bildes mit einer Bittiefe *B* der Wert 2^{B-1} subtrahiert wird. Für vorzeichenbehaftete Eingangswerte entfällt dieser Schritt. In beiden Fällen stehen anschließend die Eingangswerte im Intervall $[-2^{B-1},2^{B-1}-1]$ zur weiteren Verarbeitung bereit. Ebenfalls optional ist die Durchführung einer Farbraumtransformation, um Helligkeits- und Farbwerte zu dekorrelieren und damit die Kompressionseffizienz der nachfolgenden Komponenten weiter zu steigern. Wird eine Farbraumtransformation gewählt, muss bereits hier zwischen einer verlustbehafteten und einer verlustfreien Verarbeitung unterschieden werden.

- Bei verlustloser Komprimierung muss die *Reversible Color Transformation* (RCT) genutzt werden. Hierbei handelt es sich um eine sog. *Integer-to-Integer*-Transformation, die auf Ganzzahlen beruht und dadurch reversibel ist.

- Bei verlustbehafteter Kompression kann ebenfalls die RCT genutzt werden. Alternativ steht die *Irreversible Color Transform* (ICT) zur Verfügung, die einer RGB zu YC_bC_r-Transformation entspricht.

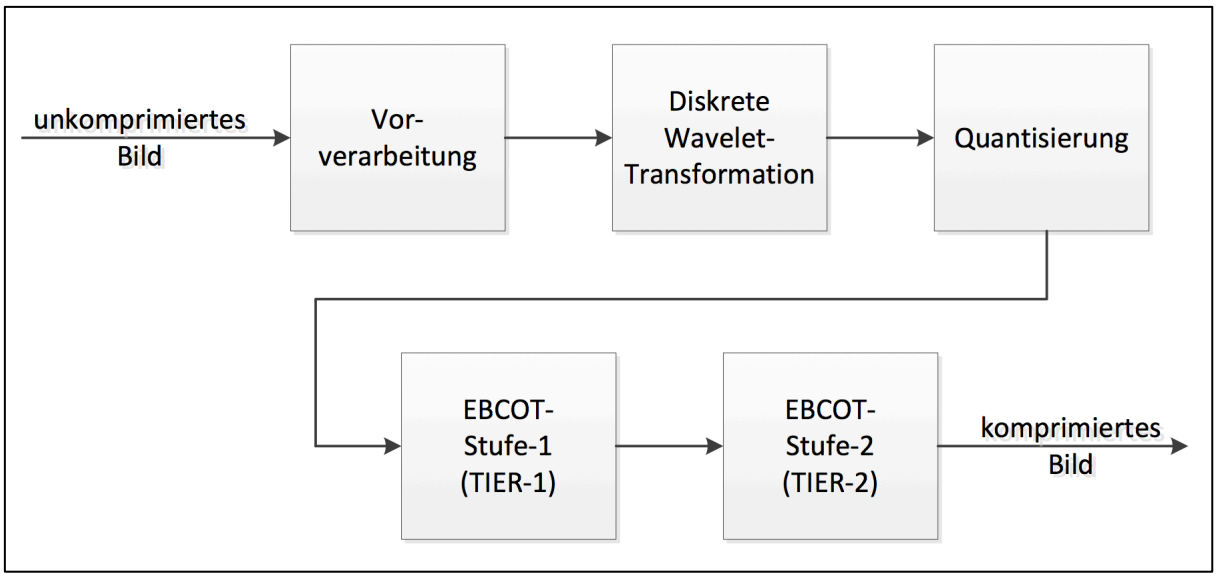

Abbildung 2: Verarbeitungsschritte zur JPEG 2000-Komprimierung

Diskrete Wavelet-Transformation:

Auf eine formale Definition für die Diskrete Wavelet-Transformation (DWT) soll an dieser Stelle verzichtet werden. Der interessierte Leser möge folgenden Literaturverweis als Einstieg in die Grundlagen zur Wavelet-Theorie nutzen [123].

Für die Anwendung innerhalb von JPEG 2000 ist in diesem Kontext wichtig, dass sog. *Wavelet-Filter* existieren, die in einer Filterbank zusammengefasst sind. Eine Filterbank besteht jeweils aus einem Filter mit Tiefpasseigenschaften $h_0(n)/g_0(n)$ und einem Filter mit Hochpasseigenschaften $h_1(n)/g_1(n)$. Im Encoder werden sie als Analyse-Filterbank ($h_0(n)$, $h_1(n)$), im Decoder als Synthese-Filterbank ($g_0(n)$, $g_1(n)$) bezeichnet. Die Wahl der sog. *Filterkoeffizienten* ist ein entscheidender Faktor für die Effizienz der nachgeschalteten Komprimierung (vgl. [1] und [113]). Im für diese Arbeit relevanten Teil 1 des JPEG 2000-Standards stehen zwei Filterbänke zur Auswahl, deren Filterkoeffizienten in Tabelle 1 und Tabelle 2 dargestellt sind.

Die auf Ganzzahlen basierende Filterbank (vgl. Tabelle 2) ist reversibel und somit für die mathematisch verlustlose Komprimierung geeignet. Eine höhere Kompressionseffizienz, die jedoch nicht verlustlos reversibel ist, kann durch eine Filterbank (vgl. Tabelle 1) erreicht werden, die auf Gleitkommazahlen basiert [96]. Innerhalb von JPEG 2000 wird eine zweidimensionale DWT angewendet, was nachfolgend beispielhaft an einem Eingangsbild mit einer räumlichen Auflösung von 256x256 Pixeln betrachtet wird (vgl. Abbildung 3-a). Zunächst wird jede Spalte des unkomprimierten Bildes mit einer gewählten Filterbank in vertikaler Richtung prozessiert. Der Filter mit Tiefpasseigenschaft $h_0(n)$ eliminiert dabei die hochfrequenten Anteile, wodurch eine Version des Eingangsbildes entsteht, die weniger Detailschärfe zeigt. Entsprechend der Bildhöhe des unkomprimierten Eingangsbildes entstehen hier 256 Wavelet-Koeffizienten. In einem nachgeschalteten Downsampling-Schritt

Tabelle 1: Analyse- und Synthese-Filterkoeffizienten der Daubechies-Feauveau 9-tap/7-tap Wavelet-Filterbank nach [96]

Tap	**Analyse**		**Synthese**	
	Tiefpass $h_0(n)$	**Hochpass $h_1(n)$**	**Tiefpass $g_0(n)$**	**Hochpass $g_1(n)$**
0	+0.062949018236360	+1.115087052457000	+1.115087052457000	+0.602949018236360
±1	+0.266864118442875	-0.591271763114250	+0.591271763114250	-0.266864118442875
±2	-0.078223266528990	-0.057543526228500	-0.057543526228500	-0.078223266528990
±3	-0.016864118442875	+0.091271763114250	-0.091271763114250	+0.016864118442875
±4	+0.026748757410810			+0.026748757410810

Tabelle 2: Analyse- und Synthese-Filterkoeffizienten der LeGall 5-tap/3-tap Wavelet-Filterbank nach [96]

Tap	**Analyse**		**Synthese**	
	Tiefpass $h_0(n)$	**Hochpass $h_1(n)$**	**Tiefpass $g_0(n)$**	**Hochpass $g_1(n)$**
0	6/8	1	1	6/8
±1	2/8	-1/2	1/2	-2/8
±2	-1/8			-1/8

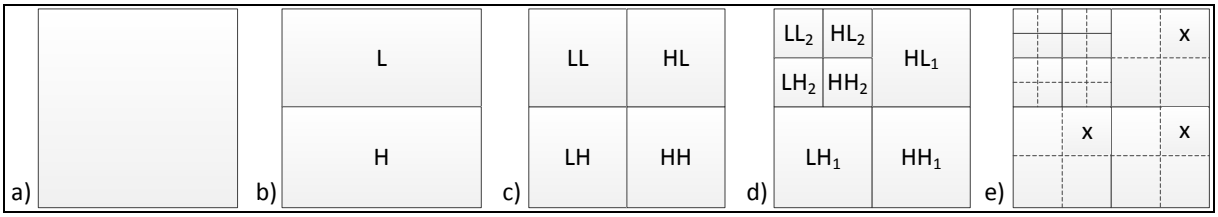

Abbildung 3: Einzelschritte bei der Anwendung der zweidimensionalen Wavelet-Transformation (a-c), Subband-Anordnung nach zweifacher Ausführung der zweidimensionalen DWT (d), Einteilung von Code-Blöcken zu Precincts (e)

wird jeder zweite Wavelet-Koeffizient verworfen. Durch die Eigenschaft *perfect reconstruction* (PR) der verwendeten Filterbänke [96,123] können die verworfenen Koeffizienten während der Synthese wieder verlustfrei zurückgewonnen werden. Entsprechend stehen nach dem Downsampling 128 Wavelet-Koeffizienten zur Verfügung, die in einem sog. *Subband*, in diesem Fall dem L-Subband abgelegt werden (vgl. Abbildung 3-b). Parallel dazu wird der Filter mit Hochpasseigenschaft prozessiert. Durch Eliminierung der niederfrequenten Anteile entsteht eine Darstellung, die vor allem Kanten, Texturen und Details zeigt. Auch hier findet nach Anwendung der Filterung ein Downsampling statt, wobei jeder zweite Wavelet-Koeffizient verworfen wird. Entsprechend stehen 128 Wavelet-Koeffizienten zur Verfügung, die im sog. *H-Subband* abgelegt werden (vgl. Abbildung 3-b).

Zur Vollständigkeit sei hier noch angemerkt, dass die DWT auch auf Eingangsdaten angewendet werden kann, die eine ungerade Bildhöhe bzw. Bildbreite aufweisen. In diesem Fall beinhaltet entweder das L-Subband oder das H-Subband einen zusätzlichen Koeffizienten, wodurch während der Synthese die ursprüngliche Repräsentation wieder hergestellt werden kann. Welches Subband den zusätzlichen Koeffizienten beinhaltet, ist in JPEG 2000

von der Position des Eingangsbildes in Abhängigkeit vom verwendeten Canvas-Koordinatensystem abhängig. Dieser Sonderfall wird hier nicht weiter betrachtet. Der interessierte Leser findet hierzu in [96] weiterführende Informationen.

In einem zweiten Schritt dienen die oben berechneten Wavelet-Koeffizienten als Eingangsdaten für eine weitere, horizontale Anwendung der DWT, was zu einer erneuten Aufteilung in nieder- und hochfrequente Anteile führt. Analog zur vertikalen Filterung wird anschließend ein Downsampling durchgeführt. Als Resultat der einfachen Ausführung einer zweidimensionalen DWT stehen vier sog. *Subbänder* — im konkreten Beispiel mit je 128x128 Werten — zur Verfügung (vgl. Abbildung 3-c). Jedes Subband erhält gemäß seiner Filterung während der zweidimensionalen DWT ein entsprechendes Label:

- *LL-Subband* – Durchlief in beiden Filterungsschritten den Tiefpass.
- *LH-Subband* – Durchlief in der vertikalen Filterung den Tiefpass und in der horizontalen Filterung den Hochpass.
- *HL-Subband* – Durchlief in der vertikalen Filterung den Hochpass und in der horizontalen Filterung den Tiefpass.
- *HH-Subband* – Durchlief in beiden Filterungsschritten den Hochpass.

Eine weitere Ausführung der zweidimensionalen DWT kann anschließend auf dem LL-Subband des letzten Berechnungsschrittes wiederholt werden (vgl. Abbildung 3-d).

Auf Grund der vorgestellten Verfahren lassen sich die räumlichen Dimensionen der einzelnen Skalierungsstufen einfach ermitteln. Sei dim_0 die Bildbreite und Bildhöhe des Eingangsbildes, so gilt für die weiteren Auflösungsstufen N_L: $dim_{N_L}(N) = \left\lceil \frac{dim_0}{2^N} \right\rceil$. Abbildung 4 zeigt die vierfache Anwendung der zweidimensionalen DWT. Es ist zu erkennen, dass die verschiedenen Auflösungsstufen bereits die erste Skalierungsvariante innerhalb des JPEG 2000-Standards ermöglicht. Hierdurch kann später wahlweise eine der vorliegenden Auflösungsvarianten aus dem komprimierten Datenstrom extrahiert werden. Zur Kompression selbst trägt die DWT nicht bei, erreicht jedoch eine Informationskonzentration in den LL-Subbändern.

Während der Wavelet-Synthese werden die entsprechenden Synthese-Filterbänke $g_0(n)$ und $g_1(n)$ verwendet, um aus den einzelnen Subbändern (z. B. LL_4, LH_4, HL_4 und HH_4 in Abbildung 4) wieder Bildrepräsentation mit einer höheren räumlichen Auflösung (LL_3) zu erstellen.

Quantisierung:

Die Quantisierung wird separat für jedes Subband ausgeführt und kommt nur im verlustbehafteten Modus zum Einsatz. Die anzuwendenden Schrittgrößen für die Quantisierung können in jedem Subband beliebig gewählt werden. Eine Gewichtung der Subbänder ist dabei ebenso

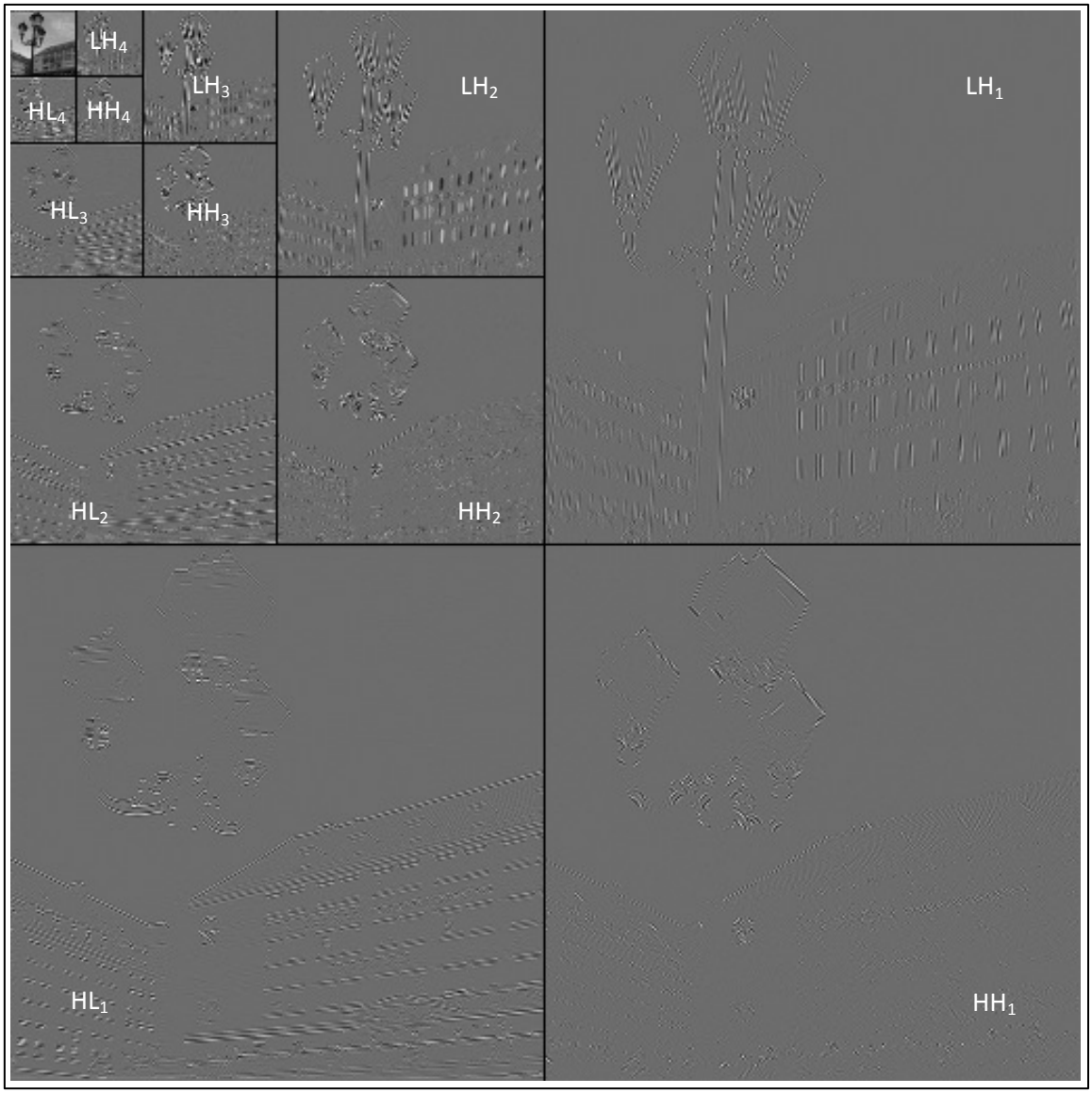

Abbildung 4: Vierfache Ausführung der zweidimensionalen DWT (erstellt mit [37])

möglich wie die Verwendung von aufgenommenen Messwerten der menschlichen Wahrnehmung bei der Betrachtung von Bildern (Human Visual System - HVS). Die eigentliche Kompression wird nach einer Quantisierung im nachfolgenden Schritt realisiert.

EBCOT-Stufe-1 (TIER-1):

Motiviert durch die Anforderung, dass ein JPEG 2000-Datenstrom eine hohe Skalierbarkeit aufweisen muss, wird im *Embedded Block Coding with Optimized Truncation*-Algorithmus (EBCOT) jedes Subband einzeln komprimiert. Hierdurch werden vorhandene Korrelationen zwischen den Subbändern zwar nicht zur Steigerung der Komprimierungseffizienz genutzt. Dem gegenüber steht jedoch der Vorteil, dass die einzelnen Subbänder besser in den resultierenden Datenstrom eingebettet werden können, parallel decodierbar sind und eine effiziente Ratenkontrolle ermöglichen. Zudem werden die Subbänder in sog. *Blöcke* bzw. *Code-Blöcke* (engl. *code-blocks*) fester Größe (z. B. 32x32) unterteilt, die ebenfalls

unabhängig voneinander encodiert werden. Die Begrifflichkeiten und deren Zusammenhang sind in Abbildung 5 gezeigt.

Ziel der ersten, hier beschriebenen EBCOT-Stufe ist es, die benötigte Datenmenge für die Darstellung der Koeffizienten mittels eines Entropie-Encoders zu minimieren. Stufe-1 beinhaltet hierzu den sog. *Context-Adaptive Binary Arithmetic Coder (CABAC)*, der alle Code-Blöcke, beginnend mit den höchstwertigen Bits bis zu den niedrigstwertigen Bits, verarbeitet. Hierzu wird das Konzept der sog. *Bitplanes* verwendet, die eine andere Präsentationsform der quantisierten Wavelet-Koeffizienten darstellen (vgl. Abbildung 5). Bitplane 1 enthält das Bit mit dem höchsten Stellenwert (engl. *Most Significant Bit - MSB*) aller Koeffizienten des aktuellen Blocks, Bitplane 2 das Bit mit dem zweithöchsten Stellenwert usw. Entsprechend enthält Bitplane *n* das Bit mit dem niedrigsten Stellenwert (engl. *Least Significant Bit - LSB*) aller Wavelet-Koeffizienten des aktuellen Blocks. Jede Bitplane wird innerhalb der EBCOT-Stufe-1 in verschiedenen Codierungspässen durchlaufen. Beinhaltet dabei Bitplane 1 keine logischen ‚1'-Werte, werden keine Daten für die Bitplane erzeugt und mit Bitplane 2 fortgefahren, entsprechend bis die ersten Daten für einen Code-Block erzeugt wurden. Ab diesem Zeitpunkt werden alle noch nachfolgenden Bitplanes des aktuellen Blocks encodiert. Ziel dabei ist es, signifikante Bits zu identifizieren. Zu Beginn der Ausführung sind alle Koeffizienten insignifikant. Ein Codierungspass umfasst drei Sub-Codierungspässe:

1. *Significance Propagation Pass* – In diesem Pass werden insignifikante Koeffizienten encodiert, die auf Grund ihrer Nachbarschaft eine hohe Wahrscheinlichkeit aufweisen, signifikant zu werden. Hierzu werden die jeweiligen Nachbarkoeffizienten berücksichtigt. Ist einer der Nachbarn bereits signifikant (Wert=1), wird der aktuelle Koeffizient in diesem Durchlauf berücksichtigt und sein Status je nach eigenem Wert aktualisiert.

1. *Refinement Pass* – Die Werte bereits signifikanter Koeffizienten werden in diesem Pass encodiert.

2. *Clean-up Pass* – Codiert die noch nicht berücksichtigten Koeffizienten des aktuellen Code-Blocks.

Nach dem Durchlauf der Sub-Codierungspässe für jeden Code-Block stehen die komprimierten Daten zur Verfügung. Abbildung 6 zeigt hierzu exemplarisch, wie der Zustand nach der Durchführung der EBCOT-Stufe-1 aussehen könnte. Das Beispiel zeigt, dass die ersten Bitplanes verschiedener Subbänder (Ausnahme: LL2) keine logische ‚1' beinhalten und deshalb keine Daten für diese Bitplane erzeugt wurden. Weiter zeigt Abbildung 6 die Unabhängigkeit der einzelnen Pässe. Sub-Codierungspässe können später bei der Erstellung des Datenstroms verworfen werden. Dieser Mechanismus bietet die Skalierungsmöglichkeit der Qualität innerhalb von JPEG 2000.

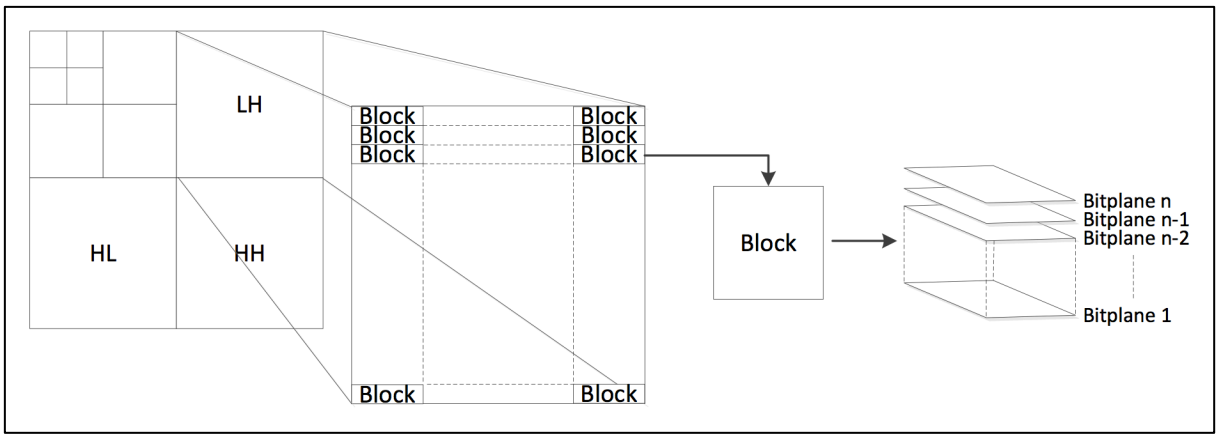

Abbildung 5: Darstellung von Subband, Block (Code-Block) und Bitplane

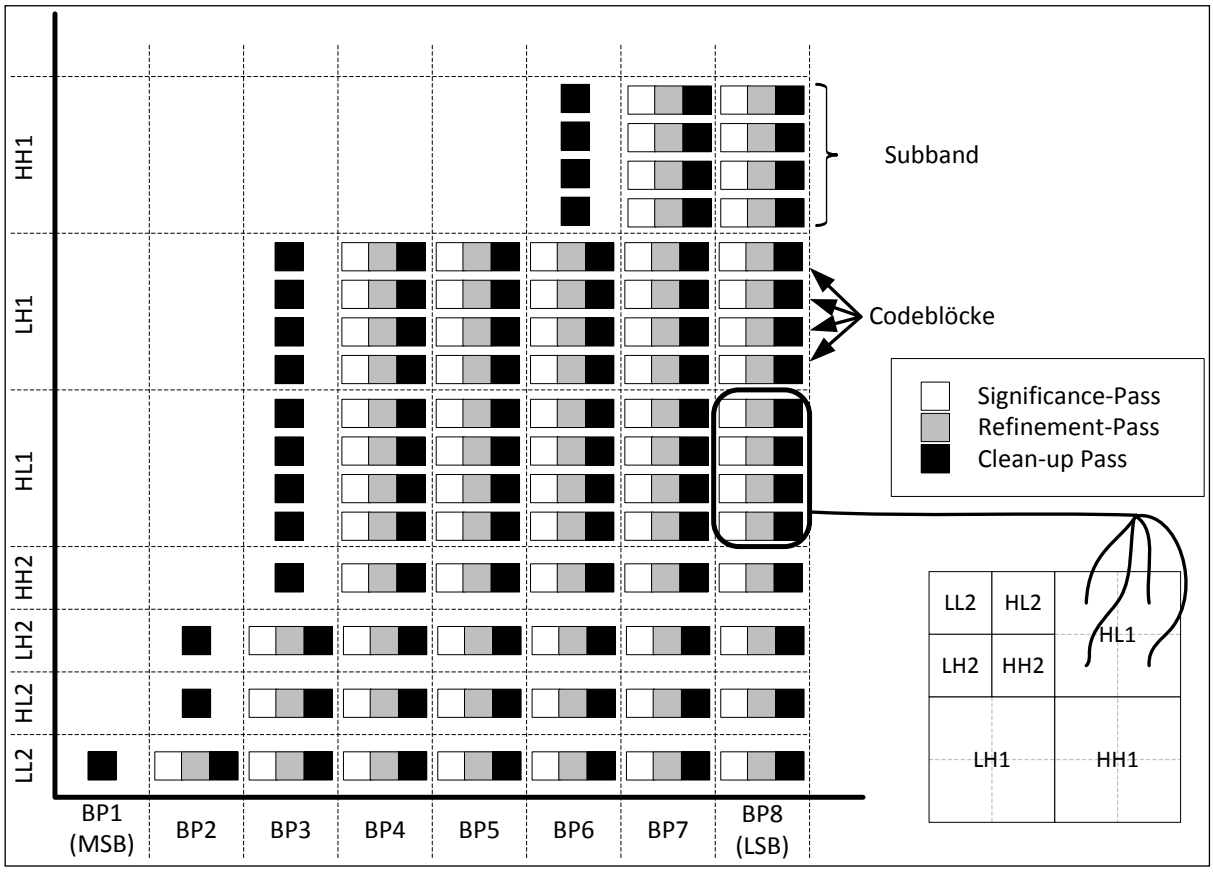

Abbildung 6: Anwendung der Codepässe auf die Subbänder der Wavelet-Koeffizienten für jede Bitplane (BP) und den Subbändern (LLx, HLx, LHx und HHx) nach [96], Detailbeschreibung im Fließtext

EBCOT-Stufe-2 (TIER-2):

In der letzten Stufe des JPEG 2000-Encoders — der EBCOT-Stufe-2 — werden die generierten Daten abschließend zu einem sog. *skalierbaren Datenstrom* (engl. *scalable bitstream)* zusammengefügt. Hierzu werden weitere Organisationsstrukturen eingeführt:

- Bezirke (engl. *Precincts*) – Ein Bezirk ist eine Gruppe von Code-Blöcken aller Subbänder einer bestimmten Auflösung (vgl. Abbildung 3-e).

- (Qualitäts-) Schichten (engl. *Layers*) – Die Ergebnisse aufeinanderfolgender Pässe eines Code-Blocks werden in einer Qualitätsschicht zusammengefasst. Die Anzahl der (Sub-) Codierungspässe, die zu einer Qualitätsschicht beitragen, variiert und wird maßgeblich durch den Encoder bestimmt. Zur Generierung von JPEG 2000-Bildern mit einer bestimmten Zielbitrate werden hierzu üblicherweise Codierungspässe verworfen.

- Pakete (engl. *Packets*) – Komprimierte Daten, die zu einer bestimmten Kachel (engl. *Tile*), (Farb-) Komponente, Auflösung, Qualitätsschicht und Precinct (Position) gehören, werden in einem Paket zusammengefasst. Hierdurch stehen vier verschiedene Pakettypen zur Verfügung: (i) Auflösung (R=Resolution), (ii) Qualität (L=Layer), (iii) Komponente (C=Component) und (iv) Position (P).

Diese Pakete können anschließend durch die sog. *Progressionsreihenfolge* (engl. Progression-Order) innerhalb der EBCOT-Stufe-2 unterschiedlich angeordnet werden.

Nach [41] stehen in Part 1 des JPEG 2000-Standards folgende Varianten zur Verfügung:

1. *Resolution-Layer-Component-Position Progression (RLCP)* – Typischer Anwendungsfall ist die Auflösungsskalierung, die es erlaubt, Auflösungspakete (R) inkl. der entsprechenden LCP Pakete zur verwerfen. Hierdurch erhält ein Decoder die Daten für eine Variante des Bildes in kleinerer Auflösung.

2. *Layer-Resolution-Component-Position Progression (LRCP)* – Hierdurch kann ein Decoder sehr einfach die Qualität sukzessive verbessern.

3. *Resolution-Position-Component-Layer Progression (RPCL)* – Ebenfalls für die Anwendung der Auflösungsskalierbarkeit. Zusätzlich werden alle Pakete eines Precincts nacheinander im Codestream abgelegt.

4. *Position-Component-Resolution-Layer Progression (PCRL)* – Für den Anwendungsfall, dass die Qualität in einem bestimmten Bereich des Bildes sukzessiv verbessert werden soll.

5. *Component-Position-Resolution-Layer Progression (CPRL)* – Wird verwendet, wenn alle verfügbaren Daten für eine (Farb-) Komponente abgerufen werden sollen, bevor Daten für die weiteren (Farb-) Komponenten geliefert werden.

Unabhängig von der gewählten Progressionsreihenfolge gilt innerhalb von JPEG 2000 die Vorgabe, dass die Daten der Codierungspässe für jeden Code-Block in der Reihenfolge vom MSB zum LSB im resultierenden Datenstrom abgelegt sein müssen. Mit anderen Worten müssen die Daten eines bestimmten Code-Blocks für die Bitplane 1 vor den Daten für die Bitplane 2 im Datenstrom abgelegt sein.

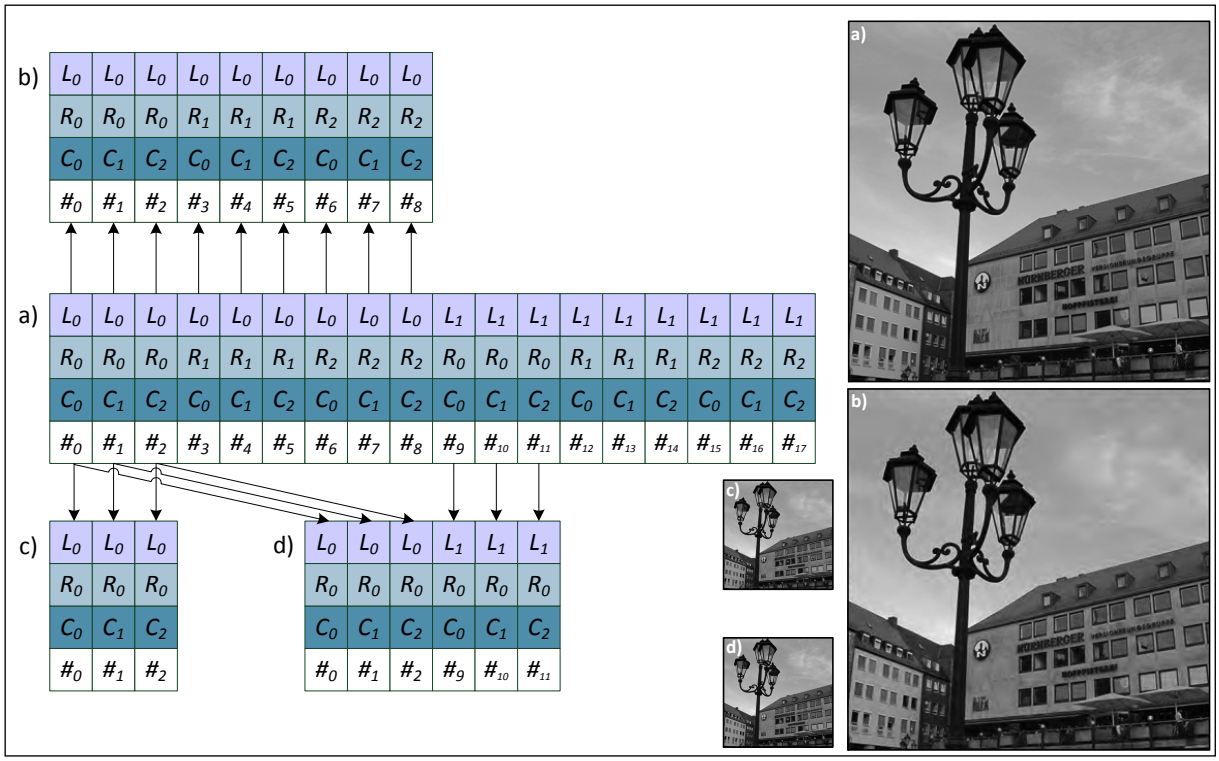

Abbildung 7: JPEG 2000-Bild mit Progressionsreihenfolge LRCP und abgeleitete Varianten a) 18 Paketen ($\#_0$-$\#_{17}$), zwei Qualitätsschichten (L_0 und L_1), drei Auflösungsstufen (R_0-R_2) und drei Farbkomponenten (C_0-C_2). Dabei ist b) abgeleitete, neue Variante mit reduzierter Qualität (L_0), c) abgeleitete, neue Variante mit reduzierter Qualität und Auflösung d) abgeleitete, neue Variante mit verringerter Auflösung (R_0). Dazu entsprechend die decodierten Bilder der abgeleiteten Varianten

Abbildung 7 zeigt ein Beispiel eines resultierenden, skalierbaren Datenstroms einer JPEG 2000-Datei (Abbildung 7-a) nach der LRCP-Progressionsreihenfolge sowie neu zusammengestellte Varianten der originalen Datei (Abbildung 7-b-Abbildung 7-d). Hierbei ist zu beachten, dass die Varianten mittels einer Neukomposition der vorhandenen Pakete erstellt werden können, ohne das vorhandene Bild decodieren zu müssen. So zeigt bspw. Abbildung 7-d eine in Auflösung und Qualität stark reduzierte Variante der originalen JPEG 2000-Datei.

Obwohl JPEG 2000 ursprünglich als Einzelbildkompressionsverfahren definiert wurde, wird es auch für die progressive Kompression von Videodaten eingesetzt. Tabelle 3 zeigt eine Auswahl von verwendeten Datenraten für JPEG 2000, wenn es für Videodaten verwendet wird. Für viele Jahre war die Decodierung der JPEG 2000-Dateien auf Standard-PCs die größte Herausforderung, da der Codec hohe Rechenleistung für das Codieren und Decodieren erfordert. Heute stehen mehrere Technologien zur Verfügung, um die JPEG 2000-Decodierung beschleunigen zu können. Dies sind u. a. Mehrkernprozessoren, dedizierte Beschleunigerkarten [2,74,93] sowie Grafikkarten [5]. Hierdurch kann eine flüssige Echtzeitausspielung erreicht werden, wenn die Daten schnell genug an die entsprechenden Recheneinheiten geliefert werden. Hier kann sich allerdings ein neuer Engpass durch zu langsame Datenträger und/oder Schnittstellen ergeben.

Tabelle 3: Datenraten bei der professionellen Verwendung von JPEG 2000

Anwendungsgebiet	**Räumliche Auflösung**	**Maximale Datenrate**	**Kompressions-faktor**
Digital Cinema Package (DCP)	2048x1080 (2K)/ 4096x2160 (4K)	31,25 MB/s	~7,3:1 (2K) ~14,6:1 (4K)
Intermediate Archive Package (IAP)	2K/4K	62,5 MB/s	~3,65:1 (2K) ~7,3:1 (4K)
Master Archive Package (MAP)	beliebig	unbegrenzt	~2:1

2.1.3 H.26x, MPEG-x und H.264 SVC

Im Bereich der Videocodierung existieren bereits eine Vielzahl skalierbarer Kompressionsverfahren. Zwar lieferten die frühesten Verfahren, *H.261* [46] von der *International Telecommunication Union* (ITU) sowie MPEG-1 von der *International Organization for Standardization/International Electrotechnical Commission* (ISO/IEC), noch keine Skalierbarkeit, jedoch beinhalteten die Folgestandards *H.262/MPEG-2* [47], *H.263* [48] sowie *MPEG-4 Visual* [39] bereits ein gewisses Maß an Skalierbarkeit. Hierdurch wurde es bspw. möglich, Subvideos mit reduzierter zeitlicher und räumlicher Auflösung oder Qualität zu extrahieren. In 2003 wurde die nächste Generation eines Videocodecs — zunächst ohne skalierbare Erweiterung — vorgestellt. Das sog. *H.264-Format* wurde durch den Zusammenschluss (*Joint Venture Team* – JVT) von der ITU sowie der *Moving Picture Expert Group* (MPEG)-Visual verabschiedet und eignet sich sowohl für mobile Anwendungen wie für die Bereiche Consumer und (Semi-) Professional. Besonders die Anforderungen der letztgenannten Bereiche konnten durch die spätere Erweiterung *High Profile* [49] abgedeckt werden. Zusätzliche Erweiterungen seit 2003 sind das sog. *Multi View Coding* (MVC), *Reconfigurable Video Coding* (RVC) und *Scalable Video Coding* (SVC). MVC adressiert dabei den Trend der 3D Anwendungen, bei dem mehr als eine Ansicht einer Szene aufgenommen und versendet (distributiert) wird. Für stereoskopische Wiedergaben, die eine 3D-Brille beim Betrachter erfordern, werden zwei Ansichten – jeweils für das linke und das rechte Auge – distributiert. Für sog. *autostereoskopische Multi-View-Displays* werden üblicherweise fünf bis neun Ansichten versendet. Dabei werden die Redundanzen zwischen den Ansichten zur Steigerung der Kompressionseffizienz genutzt, indem lediglich die Differenzen benachbarter Bilder encodiert werden. RVC bietet Ansätze zur schnelleren Erstellung neuer Kompressionsalgorithmen durch die Definition einer Werkzeug-Bibliothek (engl. *Tool-Library*). Ziel von SVC ist die Erstellung eines qualitativ hochwertigen Videostroms, der ein oder mehrere Teildatenströme beinhaltet, der beim Decodieren in seiner Komplexität und Qualität sowie in der benötigten Datenmenge mit H.264 AVC vergleichbar ist [97]. Weiterhin adressiert SVC verschiedene Defizite seiner Vorgänger und erfüllt dadurch die folgenden Anforderungen: (i) Die Codiereffizienz ist vergleichbar zu nicht-skalierbaren Verfahren, (ii) die Komplexität des

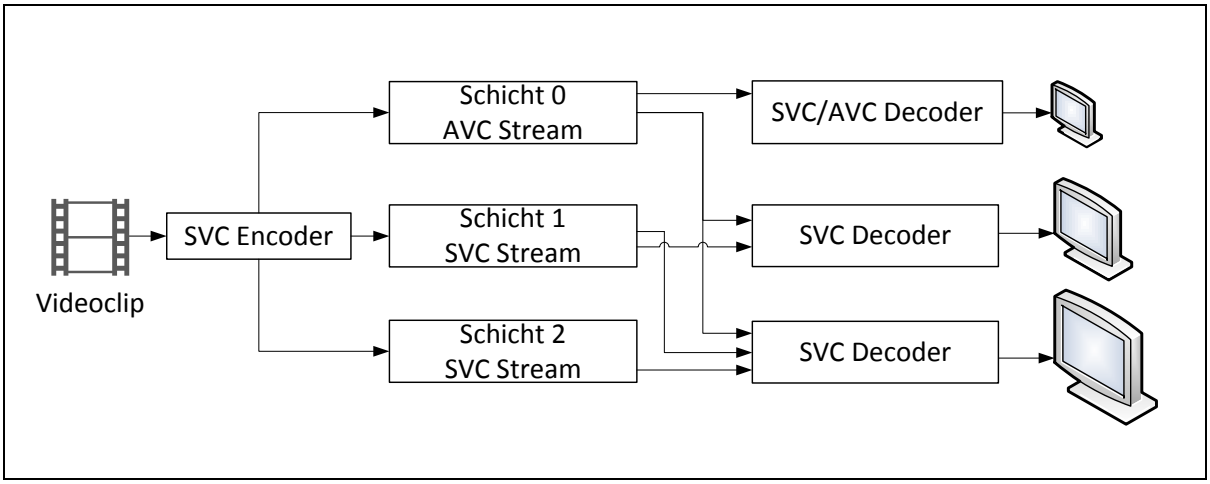

Abbildung 8: Codierung und Decodierung eines H.264 SVC-Videoclips nach [90]

Decoders für Spatio-Temporal- und Auflösungsskalierbarkeit ist nur minimal komplexer im Vergleich zu einem nicht-skalierbaren Verfahren, (iii) es werden Zeit-, Auflösungs- und Qualitätsskalierung sowie (iv) eine einfache Datenstrom-Adaption nach der Encodierung unterstützt.

Hierdurch bietet H.264 SVC die Möglichkeit, verschiedene Varianten eines Videos in unterschiedlicher Größe, Qualität sowie zeitlicher Auflösung in einen Datenstrom einzubetten. Typischer Anwendungsfall ist dabei ein Video, das über verschiedene Distributionswege an Endgeräte mit unterschiedlichsten Bildschirmgrößen und Prozessorleistungen gesendet werden muss.

Neben der Nutzung eines skalierbaren Videocodecs ist die Erstellung von vielen H.264 AVC-Videos, die über verschiedene Distributionswege an die entsprechenden Abnehmer gesendet werden (*Simulcast*), denkbar. Da hierbei allerdings das gleiche Video mehrfach encodiert wird, besteht in den komprimierten Datenströmen eine gewisse Redundanz, die zur Effizienzsteigerung der Kompression genutzt werden kann.

Abbildung 8 zeigt das prinzipielle Verfahren von H.264 SVC. Der SVC-Encoder erstellt hier drei verschiedene Schichten, genannt *Layer*. Schicht 0 (Basisschicht, engl. *Base Layer*) ist dabei ein standardkonformer sog. *H.264 AVC-Single-Layer-Datenstrom*. Darüber hinaus erzeugt der SVC-Encoder mehrere Erweiterungsschichten (engl. *Enhancement Layer*), die ein entsprechender SVC-Decoder dazu nutzen kann, die Auflösung, die Qualität oder die Bildwiederholrate der bereits vorhandenen Basisschicht zu erhöhen. Hierzu verarbeitet der Decoder die Basisschicht und entsprechend viele Erweiterungsschichten, bis die gewünschten Abspielparameter erreicht sind oder keine weiteren Erweiterungsschichten mehr zur Verfügung stehen.

H.264 SVC bietet drei Arten der Skalierbarkeit, die bei Bedarf auch miteinander kombiniert werden können:

1. Skalierung der Bildwiederholrate (engl. *Temporal Scalability*) – Die Basisschicht liefert Informationen für eine niedrige Bildwiederholfrequenz. Die Erweiterungsschichten erhöhen sukzessive diese Frequenz. Abbildung 9 zeigt die Basisschicht mit den Bildern 0, 6, 12, usw. – die Bildwiederholfrequenz ist F_0. Erweiterungsschicht 1 beinhaltet die Bilder 3, 9, 15, usw. Durch Decodierung der Basisschicht und Erweiterungsschicht 1 wird eine Bildwiederholrate von $2F_0$ erzeugt. Erweiterungsschicht 2 beinhaltet die Bilder 1, 2, 4, 5, 7, 8, usw. Durch Decodierung der Basisschicht sowie der Erweiterungsschichten 1 und 2 ergibt sich eine Bildwiederholrate von $6F_0$. Für die Prädiktion der *P*- und *B-Frames* werden in [90] und [98] dyadische Verfahren, nichtdyadische Verfahren sowie latenzfreie Verfahren vorgestellt.

2. Skalierung der Auflösung (engl. *Spatial Scalability*) – Die Basisschicht 0 liefert Informationen für eine geringe Bildgröße. Die Erweiterungsschichten liefern die entsprechenden Daten, um die Bildgröße zu erhöhen. Für die Erstellung eines Datenstroms mit zwei Auflösungsstufen, wird aus einem Quellbild A mittels Downsampling eine entsprechend kleinere Version A' erstellt (vgl. Abbildung 10). A' dient als Quelle für Encoder 0, der die Basisschicht 0 erstellt. Der resultierende Datenstrom kann von einem H.264 AVC-Decoder decodiert werden, wodurch eine Abwärtskompatibilität gewährleistet ist. Anschließend wird das gerade erzeugte Bild der Basisschicht 0 für die noch zu erstellende Erweiterungsschicht wieder decodiert und durch entsprechende Algorithmen so vergrößert, dass die gewünschte Auflösung für die erste Erweiterungsschicht erreicht wird. Hierdurch entsteht ein Referenzbild für den Encoder 1. Somit stehen dem Encoder 1 das originale Bild A und das rekonstruierte und vergrößerte Bild A' zur Erzeugung der Erweiterungsschicht 1 zur Verfügung. Entsprechend benötigt Decoder 1 sowohl das vergrößerte Ergebnis von Decoder 0, sowie den Datenstrom der Erweiterungsschicht 1 für die komplette Rekonstruktion von Bild A1.

3. Skalierung der Qualität (engl. *Quality Scalability*) – Hierdurch werden sich ergänzende Daten in verschiedenen Schichten gekapselt, um dadurch Videos mit verschiedenen Qualitäten erstellen zu können [119]. Dabei werden verschiedene Quantisierungsparameter für jede Qualitätsschicht gewählt — somit erreicht die Qualitätsskalierung eine Verfeinerung der Bildqualität durch schrittweise Verkleinerung der Quantisierungsschritte [91]. Dieses Vorgehen stellt einen gesonderten Fall der Auflösungsskalierung dar, bei der die Auflösung zwischen zwei Schichten konstant bleibt und lediglich die Quantisierungsschrittgrößen angepasst werden. Dieser Ansatz, bei dem ein Video in einzelne Qualitätsschichten unterteilt wird, wird als *Coarse Grain Scalability* (CGS) bezeichnet. Hierbei wird die Bewegungskompensation in jeder Schicht separat betrachtet, es existiert somit eine sog. *Bewegungskompensationsschleife* (engl. *Motion-Compensation-Loop*) für jede Qualitätsschicht. Dadurch können bei CGS die Qualitäten (a) nur zwischen den vordefinierten Qualitätspunkten geschaltet werden und (b) lediglich an bestimmten Abspielpunkten (z. B. I-Frames) umgeschaltet wer-

den. Deshalb werden bei H.264 SVC noch weitere Verfahren zur Qualitätsskalierung vorgeschlagen, die alle innerhalb einer Auflösungsschicht ausgeführt werden. Hierdurch basieren alle Qualitätsschichten auf der gleichen Bewegungskompensation, was zwar die Effizienz steigert, aber auch das sog. *Drift-Problem* hervor rufen kann, bei dem die Prädiktionsschleifen im En- und Decoder nicht mehr synchron arbeiten und dadurch Kompressionsartefakte entstehen können. Zur Lösung des Drift-Problems werden in [49,91,119] Verfahren zur *Medium Grain Scalability* (MGS) und *Fine Grain Scalability* (FGS) vorgeschlagen, die das Drift-Problem minimieren und durch feinere Granularität die Effizienz der Qualitätsskalierung erhöhen.

Auch dem *High Efficiency Video Coding* (HEVC) — dem Nachfolgeverfahren von H.264 AVC — soll eine skalierbare Erweiterung aufgesetzt werden. Ein Aufruf (engl. *Call for Proposals)* soll hierzu mögliche technische Lösungen aufzeigen [28].

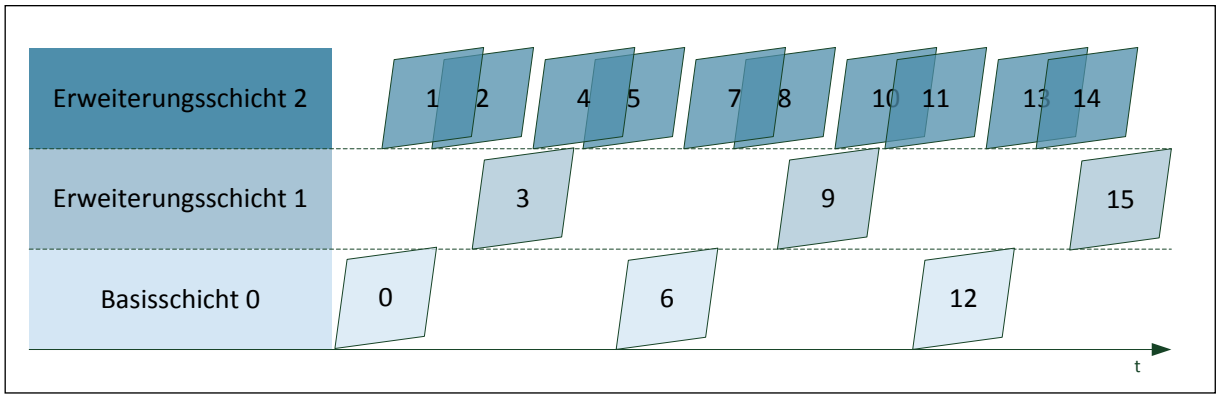

Abbildung 9: Temporale Skalierbarkeit bei H.264 SVC nach [90]

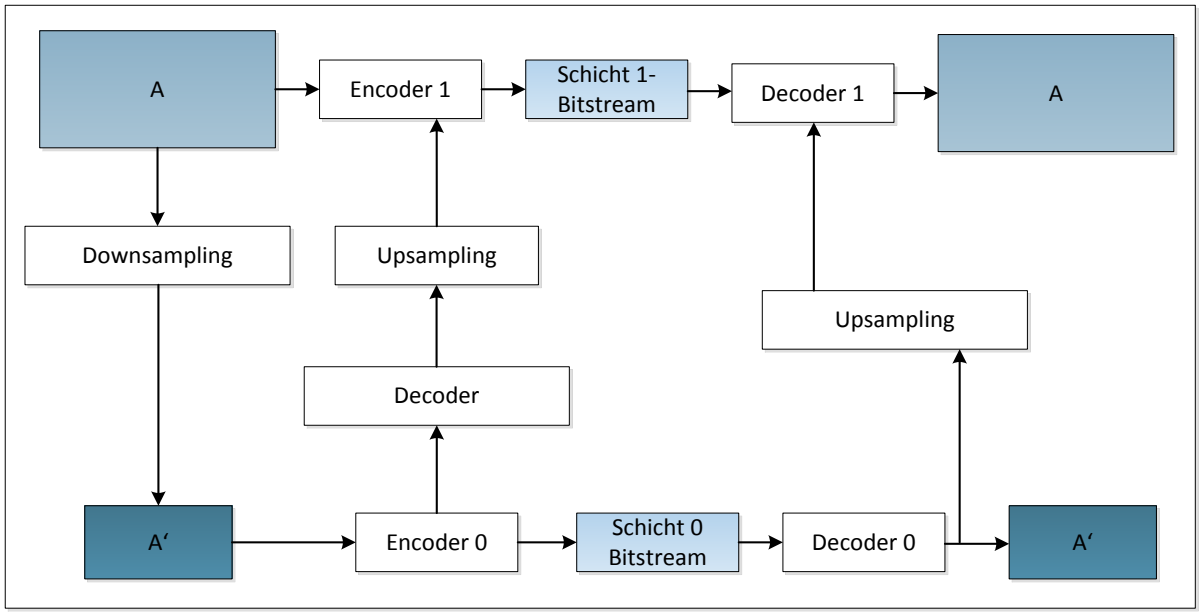

Abbildung 10: Codierung und Decodierung für die Auflösungsskalierbarkeit bei H.264 SVC nach [90]

2.1.4 Skalierbare Audioformate

Im Bereich der Audio-Codierung wird vorrangig die Qualitätsskalierung verwendet [20]. Hierbei wird ein Datenstrom so angeordnet, dass er mehrere Qualitätsschichten beinhaltet. Je mehr Qualitätsschichten decodiert werden, desto besser ist die Qualität der decodierten Audio-Datei. Das Konzept der Skalierbarkeit im Zeitbereich wurde in [3] vorgestellt. Die Erweiterung [26] erlaubt die Erstellung mehrerer Qualitätsschichten im Frequenzbereich durch eine grobe Quantisierung für die Basisschicht, die anschließend in den höheren Qualitätsschichten verfeinert wird. Dieser Ansatz wurde vor allem in der skalierbaren Erweiterung von AAC in MPEG-4 [24,40] übernommen. Das Konzept für einen feinkörnigen, skalierbaren Audio-Codec mit der Bezeichnung *Bitsliced Arithmetic Coding* (BSAC) wurde in [79] vorgestellt und ebenfalls in MPEG-4 standardisiert. Die feinkörnige Skalierbarkeit wird möglich, da der Encoder niedrige Bitslices ignorieren kann [20]. Dieser Vorgang ist deutlich feiner im Vergleich zu den oben genannten Verfahren, bei denen eine komplette Qualitätsschicht verworfen werden muss. Weiter ist eine Mono-/Stereo-Skalierung möglich, wobei ein Mono-Signal durch weitere Daten zu einem Stereo-Signal erweitert wird. Eine ausführliche Beschreibung über *MPEG-AAC Scalable* kann unter [25] gefunden werden.

Die aktuellste MPEG-Variante eines skalierbaren Audio-Codecs ist *MPEG-4 Scalable to Lossless* (SLS) [132]. Analog zu H.264 SVC besteht auch hier eine Abwärtskompatibilität zu den bestehenden, verlustbehafteten MPEG-4 AAC-Codecs. Hierzu wird in MPEG-4 SLS ein Zwei-Schichten-Konzept verwendet, bei dem die Basisschicht (engl. *Core-Layer*) einen AAC-kompatiblen Datenstrom darstellt und zusammen mit der Erweiterungsschicht im resultierenden, verlustlosen Datenstrom abgelegt wird [131]. Feinkörnige Skalierbarkeit von verlustlos bis verlustbehaftet wird bei MPEG-SLS durch die sog. *Bit-Plane-Coding*-Technik erreicht. Auch hier steht als Resultat ein Datenstrom bereit, der nahezu beliebig abgeschnitten werden kann.

Mit *OptimFROG* [21] und *WavPack* [6] stehen weiter zwei Open-Source-Varianten skalierbarer Audio-Codecs zur Verfügung. Beide Verfahren erzeugen jeweils eine Datei, die verlustbehaftete Daten abspeichert, wie auch eine Erweiterungsdatei, mit deren Hilfe eine verlustlose Variante erstellt werden kann. Je nach Bedarf decodiert der Empfänger lediglich die Basisdatei und erhält dadurch eine verlustbehaftete Repräsentation der originalen Audiodatei. Oder der Empfänger dekomprimiert beide vom Encoder erzeugten Dateien, um dadurch eine verlustfreie Kopie der Originaldatei zu generieren. Als hybrider Audio-Codec für Sprache und Musikdaten ist an dieser Stelle noch der skalierbare ITU–Audio-Codec G.718 [50] zu erwähnen, der für eine Kompression von 8 und 16 kHz abgetasteten Audiosignalen konzipiert wurde. G.718 beinhaltet fünf Qualitätsschichten, von denen höhere Schichten verworfen werden können, ohne die Decodierung der unteren Schichten zu beeinflussen. Die Basisschicht ist kompatibel zu ITU-T G.722.2 bei einer Datenrate von 12,65 kb/s.

2.2 Einflussfaktoren auf die Durchsatzleistung in einem Computersystem

Dieser Abschnitt zeigt die relevanten Komponenten der Verarbeitungskette für Mediendaten innerhalb eines Computersystems. Innerhalb dieser Kette können an unterschiedlichen Stellen Durchsatzengpässe entstehen, die — besonders bei der Forderung nach Echtzeitwiedergabe — das Gesamtsystem ausbremsen. Hierzu werden die in der Filmproduktion und Filmdistribution verwendeten Computerschnittstellen, Massenspeicher, Dateisysteme sowie Cache-Verfahren vorgestellt und anschließend die Herausforderungen und Problemstellungen bei der Verwendung der entsprechenden Komponenten identifiziert.

2.2.1 Relevante Computerschnittstellen

Dieser Abschnitt gibt einen Überblick über aktuell verfügbare und verwendete Computer-Schnittstellen im Bereich der Filmproduktion und Filmdistribution. Besonders beim Datentransfer von Videodaten sind diese Schnittstellen ein potenzieller Engpass in der Verarbeitungskette. Auch wenn neuere Generationen einer bestimmten Schnittstelle eine höhere Durchsatzleistung bieten, und der Engpass durch Erneuerung der Schnittstelle umgangen werden kann, befinden sich viele Millionen Rechner mit älteren Schnittstellen im Markt und werden produktiv verwendet. Darüber hinaus werden Videodaten in Zukunft mehr Speicherplatz — und damit einen höheren Datendurchsatz der Schnittstellen — benötigen. Gründe hierfür liegen in größer werdenden Bildschirmen mit einer steigenden Pixel-Anzahl und der damit verbundenen Codierung mit höheren Datenraten zur Minimierung von Kompressionsartefakten, Erhöhung der Bildwiederholrate aus künstlerischen Aspekten sowie der Vervielfachung der Ansichten innerhalb eines Videos, wie z. B. heute bei stereoskopischen Aufnahmen oder zukünftig bei Multi-View- oder Lichtfeld[2]-Anwendungen.

Bei *Universal Serial Bus* (USB) [120,121,122] handelt es sich um die meist verbreitete Schnittstelle zum Anschluss externer Speicher an einen Computer. Relevanz besitzen heute die Revisionen 2.0 und 3.0, deren Nettodatenraten in Tabelle 4 angegeben sind.

Als Alternative zu parallelen Schnittstellen wurde *IEEE1394* bzw. *FireWire* — ebenfalls vorrangig für den Anschluss von externen Speichern und Digitalkameras an einen Rechner — entwickelt. Ähnlich zu USB sind auch hier die Datenraten gering (vgl. Tabelle 4), die Verbreitung der Schnittstelle aber sehr hoch. Neuere Schnittstellen, wie *DisplayPort*

[2] Eine Lichtfeld-Kamera erfasst für die unterschiedlichen Bildpunkte neben der Intensität auch die Richtung der einfallenden Lichtstrahlen. Dies wird bspw. durch die Nutzung eines Kamerafeldes, also unter Verwendung mehrerer Kameras (z. B. 16-64), ermöglicht, die allesamt die identische Szene von einer minimal unterschiedlichen Position aufnehmen. Auch plenoptische Kameras eignen sich für das Aufnehmen eines Lichtfeldes mit Hilfe von Mikrolinsen, die vor dem Bildsensor angebracht sind.

Tabelle 4: Approximierte Nettodatenraten relevanter Computerschnittstelle

Schnittstelle	**Version/Konfiguration**	**Approximierte Nettodatenrate [MB/s]**
USB	2.0	40
	3.0	400
DisplayPort/Thunderbolt	1 Lane@162 MHz	165
	4 Lanes@540 MHz	2211
IEEE/FireWire	S400	29
	S3200	235
SATA	2.0 eSATA eSATAp	300
	3.0	600
PCI-E	1.0/1.1 1 Lane, 1,25 GHz	250
	3.0 32 Lanes, 4 GHz	3150

(DP) [127] und *Thunderbolt* [11] bieten bereits in der Grundkonfiguration mit einer Verbindung (engl. *Lane*) und einer Taktrate von 162 MHz genug Kapazität, um die aktuell benötigten Datenmengen für Digitales Kino oder Digitale Archivierung in Echtzeit transferieren zu können, sind aber bisher nicht weit verbreitet. Dabei gilt DP als Nachfolger von *Video Graphics Array*[3] (VGA) und *Digital Visual Interface* (DVI) und verfolgt als erste Display-Schnittstelle den Ansatz, Videodaten paketbasiert zu übertragen, wodurch höhere Datenraten bei geringerer Anzahl von Anschlüssen erzielt werden können. Thunderbolt kombiniert PCI-E (s. u.) und DP zu einer seriellen Daten-Schnittstelle, multiplext die entsprechenden Daten und versendet diese über eine *Duplex-Thunderbolt-Lane*. Ein Demultiplexer im Empfänger separiert die Daten entsprechend und sendet diese dann an die Endgeräte (entweder PCI-E oder DP) weiter. Serial ATA [76] dient rechnerintern zum Anschluss von z. B. Festplatten oder optischen Laufwerken an den Host-Bus-Adapter, wobei die Variante *eSATA* unter Nutzung des gleichen Protokolls, den Anschluss externer Geräte ermöglicht. Positiv wirkt sich dabei aus, dass die meisten externen Speicher mit SATA-Controllern bestückt sind und somit ein Übersetzungsschritt, z. B. auf USB, entfällt. *Peripheral Component Interconnect Express* (PCI-E) dient der internen Verbindung von Peripheriegeräten mit einem Hauptprozessor und löst die älteren Standards — *PCI, PCI-eXtended (PCI-X)* und *Accelerated Graphics Port* (AGP) — ab. Im Vergleich zu seinen Vorgängertechnologien bietet PCI-E höhere Datenraten, weniger Datenanschlüsse und dadurch eine kleinere Bauform. PCI-E ist seriell ausgelegt, wobei kommunizierende Geräte über einen PCI-E-Switch eine Punkt-zu-Punkt Verbindung aufbauen. Hierdurch wird die

[3] Neben dem hier erwähnten Schnittstellenstandard wird der Begriff VGA auch häufig mit der durch den Standard möglichen Grafikauflösung von 640x480 Pixeln verwendet.

Durchsatzleistung anderer PCI-E Teilnehmer nicht durch die Kommunikation zwischen zwei Geräten beeinflusst. Es existieren mehrere relevante Versionen von PCI-E, die — je nach Anzahl der Lanes — entsprechend hohe Datenraten aufweisen (vgl. Tabelle 4).

Die Herausforderungen und auftretenden Problemstellungen bei der Verwendung verfügbarer Computerschnittstellen in der Filmproduktion und Filmdistribution werden in Abschnitt 2.3 aufgezeigt.

2.2.2 Massenspeicher

Massenspeicher sind nicht flüchtige Medien, die große Datenmengen über einen längeren Zeitraum speichern. Sie unterscheiden sich elementar in der Art, wie sie die Daten speichern und auf gespeicherte Daten zugreifen. Jedes Verfahren bietet entsprechende Vor- und Nachteile in Bezug auf Anschaffungs- und Betriebskosten, Schreib-/Lese-Geschwindigkeit und Lebenszyklen. In diesem Abschnitt werden die relevanten Massenspeicher — und deren entscheidenden Eigenschaften — vorgestellt, die bei der Produktion, Postproduktion, Archivierung und Distribution von Mediendaten vorrangig zum Einsatz kommen.

Massenspeicher mit sequenziellem Zugriffsverhalten

Speicher mit Magnetbändern [81] werden seit Jahrzehnten zur Speicherung digitaler Informationen verwendet. Bei großen Datenmengen sind Magnetbänder eine kostengünstige und zuverlässige Möglichkeit, Daten über lange Zeit zu speichern. Besonders für die Erstellung von Sicherungskopien (engl. *Backups*) und in der Archivierung — u. a. im Bereich audiovisueller Medien — finden Magnetbänder noch häufig Einsatz. Eine wichtige Eigenschaft von Magnetbändern und die Zugriffsmethoden der Schreib-/Lesegeräte ist der sequenzielle Zugriff auf die Speichersektoren. Hierdurch ergibt sich, dass Daten in derjenigen Reihenfolge gespeichert werden sollten, in der sie später gelesen werden, da ansonsten ein zeitaufwendiges Umpositionieren des Magnetbandes nötig ist. Die meisten Systeme bieten eine optionale Kompression (Faktor ca. 2:1) der zu speichernden Daten an, um Speicherauslastung und Durchsatz zu erhöhen. Aktuelle Systeme, wie z. B. LTO-5, erreichen einen Durchsatz von etwa 75 MB/s (ohne Kompression), und können bis zu 1,5 Terabyte speichern [32].

Optische Speicher bezeichnen auswechselbare Medien, die durch optische Abtastung gelesen und wenn möglich beschrieben werden. Kommerziell erfolgreiche Verfahren umfassen CD/CD-ROM, (HD) DVD und Blu-ray Discs. Die Vorteile von optischen Speichern sind u. a. das berührungslose Abtasten durch einen Laser und der damit verbundene, geringe Verschleiß am Datenträger und an der Schreib-/Leseeinheit. Darüber hinaus sind die Fertigungskosten der Datenträger sehr gering. Nachteilig sind die geringen Lebenszeiten und die Fehleranfälligkeit der Datenträger, die von spürbaren Aussetzern der Bild- und/oder Tonwiedergabe bis

zum völligen Datenverlust führen. Viele optische Datenträger können in mehreren sog. *Layern* beschrieben werden, um die Speicherkapazität zu erhöhen (vgl. Tabelle 5).

Die Herausforderungen und auftretenden Problemstellungen bei der Verwendung verfügbarer Massenspeicher mit sequenziellem Zugriffsverhalten in der Filmproduktion und Filmdistribution werden in Abschnitt 2.3 aufgezeigt.

Massenspeicher mit wahlfreiem Zugriffsverhalten

Ein Speicher mit rotierenden Scheiben**,** auch Festplatte oder *Hard Disk*[4] *Drive* (HDD) genannt, ist ein nichtflüchtiger, blockbasierter Massenspeicher, der — im Gegensatz zu bandbasierten Speichern — beliebigen Zugriff auf die Daten erlaubt. Die Daten werden auf einer oder mehreren rotierenden Scheiben (engl. *Discs*), die mit einem ferromagnetischen Material versehen sind, gespeichert. Durch eine Einheit von Schreib- und Leseköpfen werden Daten auf den Massenspeicher geschrieben bzw. vom Massenspeicher gelesen. Üblicherweise besteht eine HDD aus mehreren Scheiben, um die Speicherkapazität zu erhöhen — es werden beide Oberflächen als Datenträger verwendet. Diese Scheiben werden so beschleunigt, dass sie bis zu 15000 Umdrehungen pro Minute absolvieren. Der Schreib-/Lesekopf kann parallel zu der rotierenden Bewegung der Platten über die Spuren der Scheiben positioniert werden — für jede Oberfläche (somit zwei pro Scheibe) steht ein Set von Lese- und Schreibköpfen zur Verfügung. Zwar sind Zugriffe auf die Datenblöcke eines Festplattenlaufwerkes frei wählbar, dennoch ist die Zugriffszeit vor allem durch die mechanischen Eigenschaften beschränkt:

- Die **Spurwechselzeit** definiert das Intervall, die der Arm des Schreib-/ Lesekopfes benötigt, um sich an einer entsprechenden Spur einer Festplatte zu positionieren.
- Die **Latenzzeit** entsteht dadurch, dass ein zu lesender Block nicht zwangsläufig direkt unter dem Schreib-/Lesekopf liegt, wenn dieser eine bestimmte Spur erreicht hat.

Tabelle 5: Datenraten optischer Speicher mit unterschiedlich vielen Schichten (engl. *Layer*) (SL – Single Layer, DL – Double Layer, TL – Triple Layer, QL – Quad-Layer)

Name	**Markteinführung**	**Speicherkapazität**	**Datenrate**
CD	1982	540 MB – 900 MB	0,1535 MB/s
DVD	1996	4,7 GB (SL) 8,5 GB (DL)	1,385 MB/s
HD DVD	2005	15 GB (SL) 30 GB (DL)	4,57 MB/s
Blu-ray Disc	2006	25 GB (SL) 50 GB (DL) 100 GB (TL) 128 GB (QL)	4,5 MB/s

[4] Der Begriff gibt an, dass die Scheiben aus einem nicht-flexiblen Material bestehen. Im Gegensatz hierzu stehen *Floppy-Disks*, deren Speicherplatten aus flexiblem Material besteht.

Festplatten mit rotierenden Scheiben sind — im Vergleich zu anderen Komponenten wie CPU und RAM — zu einem Flaschenhals in der Verarbeitungskette geworden. Besonders wenn logisch zusammenhängende Daten (z. B. eine Datei) auf mehreren räumlich getrennten Bereichen innerhalb einer Festplatte verteilt werden (z. B. wegen starker Fragmentierung der Festplatte), müssen die Schreib-/Leseköpfe entsprechend neu positioniert werden. Spurwechsel- und Latenzzeit summieren sich somit zu einem stark bremsenden Faktor beim Lesen und Schreiben. Diese Effekte können durch verschiedene Cache-Speicher reduziert werden:

- Cache im Betriebssystem – Hierdurch sollen die unterschiedlichen Schreib-/Lesegeschwindigkeiten von Massenspeicher und restlicher Computerarchitektur voneinander entkoppelt werden.

- Cache innerhalb der Festplattenelektronik – diese dient der Entkopplung der Schnittstellentransferrate und der statischen Transferrate der Schreib-/Leseköpfe. Zusätzlich werden in diesem Cache Lesedaten gespeichert, die während der Suchzeit zu einem bestimmten Block auf einer Spur gelesen werden. Grund für diese Speicherung ist die Annahme, dass diese Daten später sowieso angefragt werden, nachdem der Anfang einer Speicheradresse gefunden wurde. Durch das Caching kann der Vorgang beschleunigt werden.

Aktuell auf dem Markt verfügbare Festplatten mit einer Bauform von 3,5“ und einer typischen Kapazität von 4 Terabyte zeigen durchschnittliche Durchsatzraten beim Lesen von rund 160 MB/s [118]. Das Beschreiben einer Festplatte ist üblicherweise langsamer. Entsprechende Durchschnittswerte vom Festplatten mit den oben genannten Eigenschaften liegen bei etwa 130 MB/s. Schnittstellen zu internen Festplatten sind vielfältig und beinhalten Standardschnittstellen wie (S)ATA, IDE und SCSI. Externe Festplatten werden üblicherweise über FireWire, USB, eSATA oder Thunderbolt angeschlossen.

Die Herausforderungen und auftretenden Problemstellungen bei der Verwendung verfügbarer Massenspeicher mit wahlfreiem Zugriffsverhalten in der Filmproduktion und Filmdistribution werden in Abschnitt 2.3 aufgezeigt.

Flashbasierter Speicher

Flashbasierter Speicher bieten eine weitere Möglichkeit der persistenten Speicherung der Daten. Im Wesentlichen existieren zwei verschiedene Ausführungen, sog. *NAND*[5] und *NOR*[6] Varianten. Der Ursprung flashbasierter Speicher liegt im *Electrically Erasable Programmable Read-Only Memory* (EEPROM), welches komplett gelöscht werden muss, bevor es wiederbeschrieben werden kann. NOR-basierte Speicher erlauben das unabhängige Schreiben und Lesen eines einzelnen Maschinenwortes, NAND-basierte Flashspeicher werden in der

[5] Bezeichnet ein negiertes UND-Gatter (engl. *Negated AND*).

[6] Bezeichnet ein negiertes ODER-Gatter (engl. *Negated OR*).

logischen Untereinheit der Blöcke (ca. 4 KB) beschrieben und finden heute vornehmlich Einsatz für die Datenspeicherung in Laptops, Kameras, PDAs, MP3-Player und anderen Geräten für Multimediaanwendungen. Da die Bits bei einem Schreibvorgang nur von „1“ auf „0“, nicht aber von „0“ auf „1“, gesetzt werden können, wirkt sich bei NAND-basierten Flashspeichern der blockorientierte Zugriff als nachteilig auf die Schreibgeschwindigkeit aus. Muss mindestens ein Bit von „0“ auf „1“ gesetzt werden, ist hierfür ein Löschvorgang des kompletten Blocks nötig, bei dem alle Bits auf „1“ gesetzt werden. Entsprechend der zu speichernden Information werden anschließend Bits auf „0“ gesetzt, um die binären Daten abzuspeichern.

Solid State Drives (SSD) basieren auf flashbasierten Speichern und beinhalten keine beweglichen Teile. Dadurch sind sie stoßunempfindlich, verbrauchen weniger Strom als HDDs und bieten eine deutlich schnellere Schreib-/Lesegeschwindigkeit im Vergleich zu Festplatten. SSDs bestehen im Wesentlichen aus einem Controller und dem Speicher selbst. Der Controller besitzt einen großen Einfluss auf die Leistung des Gesamtsystems und führt neben der Adressierung der Speicherbausteine u. a. noch die Fehlerbehandlung, Bad-Block-Mapping, optionale Verschlüsselung sowie ein Schreib-/Lesecaching aus. Im Bezug auf Schnittstellen sind SSDs u. a. kompatibel zu Serial ATA, PCI-E und USB. Aktuelle SSDs zeigen durchschnittliche Durchsatzraten beim Lesen und Schreiben von rund 350 MB/s [117]. Auch wenn die Durchsatzraten deutlich über den Werten der HDDs liegen, werden klassische Festplatten noch über mehrere Jahre in der Produktion, Distribution und Archivierung von Filmen eingesetzt werden. Begründet ist dies vor allem durch den höheren Preis pro Gigabyte, der bei SSDs um einen Faktor zwölf höher ist als bei einer HDD [64].

Die Herausforderungen und auftretenden Problemstellungen bei der Verwendung verfügbarer Flashspeicher in der Filmproduktion und Filmdistribution werden in Abschnitt 2.3 aufgezeigt.

2.2.3 Hierarchische Dateisysteme

Hierarchische Dateisysteme erlauben eine Organisation der gespeicherten Dateien in Unterordnern. Die so entstehende Hierarchie bietet die Möglichkeit, Dateien nach bestimmten Regeln zu gruppieren und dadurch leichter auffindbar zu machen. Die aktuell verwendeten Dateisysteme für Massenspeicher im Consumer-Bereich sind ausschließlich hierarchisch organisiert und in der Lage, skalierbare Dateiformate zu speichern. Dennoch bieten die verschiedenen Verfahren Vor- und Nachteile, die in diesem Abschnitt erläutert werden.

File Allocation Table (FAT)

Bei dem Dateisystem *File Allocation Table* (FAT) von Microsoft handelt es sich um ein einfaches und robustes Dateisystem, das in den 1970er Jahren für Floppy-Disks entwickelt, und erst später für Festplatten adaptiert wurde. Noch heute ist FAT auf vielen Solid-State Speicherkarten und Flash-Memory-Karten das verwendete Dateisystem, von dem mehrere

Versionen existieren. Dabei zeigt der Zahlenwert im Namen die Anzahl der maximalen Tabelleneinträge in der FAT (2^{12}, 2^{16}, 2^{32}) an, d. h. FAT32 kann max. 2^{32} Speicherbereiche (Blöcke oder Cluster[7]) verwalten.

Das Dateisystem verwendet eine Indextabelle fester Größe, in der jeder Speicherbereich des Speichermediums genau einen Eintrag besitzt (vgl. Datenbereich in Abbildung 11). Dieser Eintrag gibt an, ob ein Cluster frei, belegt oder beschädigt ist. Ist ein Cluster durch eine Datei belegt, steht die Adresse des nächsten Clusters in dem Eintrag der FAT. Somit wird eine verkettete Liste aller Cluster erstellt, die Daten zu einer bestimmten Datei speichern. Der letzte Cluster einer Datei ist mit einem Sonderzeichen (*End Of File* – EOF) gekennzeichnet. Neben der FAT existiert noch das sog. *Stammverzeichnis*, in der jede gespeicherte Datei — inkl. ihres Start-Clusters in der FAT und weiteren Metadaten, wie z. B. Erstellungsdatum und Dateilänge — eingetragen ist. Verweise der FAT beziehen sich dann auf Speicherbereiche im Stammverzeichnis (vgl. Abbildung 11). Hierdurch ist es möglich, alle Dateien des Speichermediums über Stammverzeichnis, FAT und Datenbereich zu lesen.

New Technology File System (NTFS)

NTFS folgt dem Grundverständnis moderner Dateisysteme, das besagt, dass alles in einem Computersystem ein Teil einer Datei ist. Die Hauptdatei in NTFS ist die Master File Table (MFT), die eine Tabelle über alle Dateien innerhalb des Dateisystems, deren zugehörige Blöcke sowie weitere Attribute[8] und Zugriffsberechtigungen darstellen. Für jeden Dateieintrag wird Speicher in der MFT bereitgestellt und die Attribute einer Datei dort abgelegt. Die wichtigsten Parameter sind:

- *Standard Information* – beinhaltet die FAT-Standardattribute Schreib-/Lesezugriff, Erstellungsdatum, Änderungsdatum, Hard-Link-Zähler[9].
- *Dateiname* in Unicode.
- *Security Descriptor* – dient der Anwendung von Sicherheitsmechanismen.
- *Data* – Die Daten einer (kleinen) Datei oder Zeiger zu der Datei.

Kleine Dateien (weniger als 512 Bytes) und Ordner werden direkt in dem Eintrag Data der MFT abgelegt. Dateien, die größer als 512 Bytes sind und somit nicht in einen MFT-Eintrag passen, enthalten einen oder mehrere Zeiger auf diejenigen Speicherbereiche, die

[7] Sektoren (üblicherweise 2^9, 2^{10} oder mehr) auf einem Speichermedium werden zu einem Cluster zusammen gefasst.

[8] Attribute in NTFS sind Dateigröße, Datum, Dateierstellung, letzte Änderung, Freigabe, Dateityp, Dateiinhalt.

[9] Engl. *Hard Link Count* – gibt an, wie viele Verzeichnisse die Datei referenzieren.

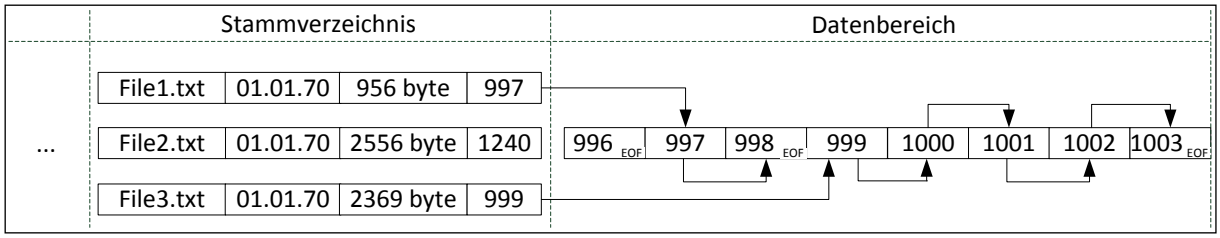

Abbildung 11: Stammverzeichnis und Datenbereich in einem FAT-Dateisystem

entsprechende Daten der Datei beinhalten (vgl. Abbildung 12). Verzeichnisse in NTFS sind ebenfalls Dateien, die eine Liste mit Name/Datei-Einträgen beinhalten.

Weitere Eigenschaften von NTFS umfassen unter anderem:

- NTFS Log – Ein *Journaling System*, das alle Operationen, die auf dem Dateisystem ausgeführt werden, protokolliert. Hierdurch ist es möglich, das Dateisystem wieder in einen konsistenten Zustand zu bringen, wenn bspw. ein gewünschter Schreibvorgang — durch einen Systemabsturz oder Hardwarefehler — nur unvollständig ausgeführt wurde.

- *Alternate Data Streams* (ADS) – Hierdurch können zusätzliche Datenstrukturen (z. B. Auflistung aller Metadaten) fest mit einer Datei verbunden werden.

- *Sparse Files* – Sind Dateien, die zu großen Teilen aus 0-Werten (zero-data) bestehen. Solche Dateien werden von Datenbanken angelegt, um Speicher für potenziell wachsende Tabellen zu reservieren und eine spätere Fragmentierung zu verhindern. Innerhalb von NTFS werden diese Dateien durch Metadaten beschrieben und lediglich die wirklichen Nutzdaten physikalisch auf dem Speichermedium gesichert. Wird eine Sparse-Datei gelesen, liefert der Controller für die Bereiche 0-Werte, die in den Metadaten dafür angegeben sind.

Extended-Dateisysteme:

Innerhalb von Betriebssystemen wie z. B. Unix oder Linux findet das *ext2*-Dateisystem starke Verbreitung. Die maximale Größe einer einzelnen Datei ist auf zwei Terabyte beschränkt und es sind bis zu 10^{18} Dateien verwaltbar. ext2 umfasst einen *Superblock*, der die Konfigurationsparameter des Dateisystems enthält. Aus Redundanzgründen werden mehrere Kopien des Superblocks an verschiedenen Stellen des Speichersystems abgelegt. Jedes Objekt wird mit sog. *Indexknoten* (engl. *Indexnode* oder kurz *Inode*) präsentiert. Die Inode-Struktur zeigt auf diejenigen Datenblöcke, die Daten und Metadaten einer Datei enthalten. Weiterhin existieren Zeiger auf zwölf spezielle Datenblöcke, deren Größe bei der Formatierung des Speichermediums festgelegt wird. Ist die Kapazität dieser Blöcke nicht ausreichend, um die Datei zu speichern, werden zusätzliche Speicherblöcke indirekt referenziert. Hierfür existiert ein Zeiger auf einen sog. *indirekten Block*, der wiederum eine Liste von weiteren Blöcken

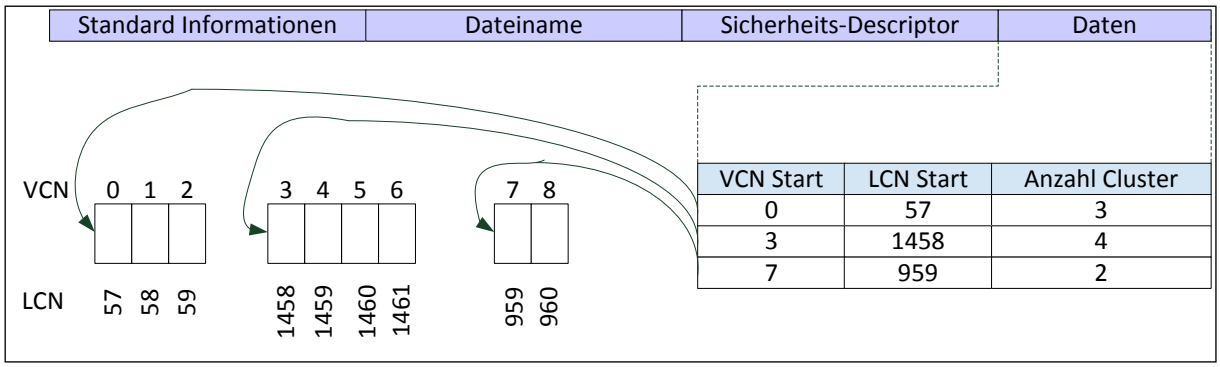

Abbildung 12: Beispieleintrag in der Master File Table (MFT) eines NTFS Dateisystems mit Logical Cluster Number (LCN) und Virtual Cluster Number (VCN)

indiziert, einen Zeiger auf einen sog. *doppelt-indirekten Block* und einen Zeiger auf einen *dreifach-indirekten Block*.

ext2 enthält kein Journaling, diese Funktion wurde erst mit *ext3* in 2001 eingeführt. Mit *ext4* [65] wurden in 2008 folgende Erweiterungen hinzugefügt: (i) sog. *Extents* sind eine Verwaltungsstruktur für Speicherbereiche von aufeinanderfolgenden Blöcken auf dem Datenträger. Ausgehend von der Annahme, dass freier Speicherplatz sequenziell auf einem Datenträger vorhanden ist, können insbesondere große Dateien effizienter durch Extends beschrieben werden. Anstatt jeden Speicherbereich der Datei im Systemkatalog zu speichern, gibt ein Extent an, mit welchem Clusterbereich er beginnt und wie viele Cluster er umfasst. (ii) Persistente Speichervorbelegung – Hierfür werden Null-Werte in eine Datei geschrieben, wenn diese erstellt wird.

Die Herausforderungen und auftretenden Problemstellungen bei der Verwendung verfügbarer Dateisysteme in der Filmproduktion und Filmdistribution werden in Abschnitt 2.3 aufgezeigt.

Virtuelle Dateisysteme und Dateisysteme im Benutzer-Modus

Reale Dateisysteme werden üblicherweise als Zusatzprogramm zum vorhandenen *Kernel-Code* erstellt. Nach der Entwicklung sind sie Teil des Kerns des Betriebssystems und werden im sog. *Kernel-Modus* — d. h. einem Modus mit besonderen Rechten — ausgeführt. Aus diesem Grund müssen sich Entwickler realer Dateisysteme mit kompliziertem Kernel-Code und den dort verwendeten Datenstrukturen auskennen. Ein Debugging der erstellten Prozeduren ist im Vergleich zu Programmcode im Applikationsbereich sehr mühsam und Fehler können einen Neustart des Systems auslösen.

Eine Alternative bieten Dateisysteme, die im Benutzermodus (engl. *User-Mode*) des Betriebssystems entwickelt und ausgeführt werden. Die Softwareentwicklung ist einfacher und Fehler führen lediglich zum Absturz des Dateisystems, nicht aber zum Absturz des Betriebssystems. Nötig für die Entwicklung von Dateisystemen im *User-Mode* ist ein Kernel-Modul, das als virtuelles Dateisystem (Virtual File System – VFS) bezeichnet wird. Das VFS

implementiert kein reales Dateisystem, gibt sich aber gegenüber aufrufenden Prozessen als vollwertiges Dateisystem aus. Anfragen an ein VFS werden an einen Prozess im *User-Mode* weitergeleitet (vgl. Abbildung 13, Dateisystem-Applikation – DSA). Üblicherweise greift die Dateisystem-Applikation auf ein reales Dateisystem zu und bereitet die darauf gespeicherten Daten — bei Bedarf mit Hilfe von weiteren Datenquellen — so auf, dass die gewünschte Funktionalität eines virtuellen Dateisystems erreicht wird.

Die Technologie *Filesystem in UserspacE* (FUSE) ist Teil des *Linux-Kernels* und steht für die Unix-Varianten Linux, FreeBSD, NetBSD, OpenSolaris und Mac OS X zur Verfügung. Vergleichbar mit FUSE ist eine Portierung für Windows, die als Open-Source-Projekt unter dem Namen *Dokan* erhältlich ist [33]. Mittlerweile existieren es eine Vielzahl von Implementierungen [102] in verschiedenen Kategorien, wie z. B. Media-Dateisysteme, Archiv-Dateisysteme oder Netzwerk-Dateisysteme.

Virtuelle Dateisysteme bieten eine alternative Möglichkeit, Prototypen von Dateisystemen mit speziellen Eigenschaften entwickeln und dadurch gewünschte Funktionsweisen realisieren und evaluieren zu können. Die Gesamtleistung eines virtuellen Dateisystems in Bezug auf Schreib-/Lesegeschwindigkeit ist allerdings geringer im Vergleich zu einem realen Dateisystem, wie in [85] genauer betrachtet wird. Das liegt im Wesentlichen an zwei grundsätzlichen Eigenschaften virtueller Dateisysteme:

1. Ein virtuelles Dateisystem erzeugt für jeden Systemaufruf vier Kontextwechsel. Zwei davon für die Übergänge zwischen Benutzer- und Kernel-Mode, zwei weitere Wechsel für die Umschaltung der Prozesse, hier (a) Applikation auf Dateisystemtreiber und (b) Dateisystemtreiber auf DSA. Besonders die Kontextwechsel der Prozesse können u. U. einen signifikanten Einfluss auf die Ausführungszeit der gesamten Anfrage ausüben. Die Kosten hängen allerdings von verschiedenen Faktoren, wie Prozessor-Typ, Auslastung oder Speicherzugriff der beteiligten Programme, ab. Letztendlich aber entsteht hier ein Zusatzaufwand im Vergleich zur Nutzung nativer Dateisysteme.

2. Die angefragten Daten werden mindestens ein weiteres Mal im Dateisystemtreiber kopiert.

Die o. g. Vor- und Nachteile realer Dateisysteme (FAT, NTFS, ext) bestehen auch bei virtuellen Dateisystemen, wenn diese auf ein reales Dateisystem zurückgreifen.

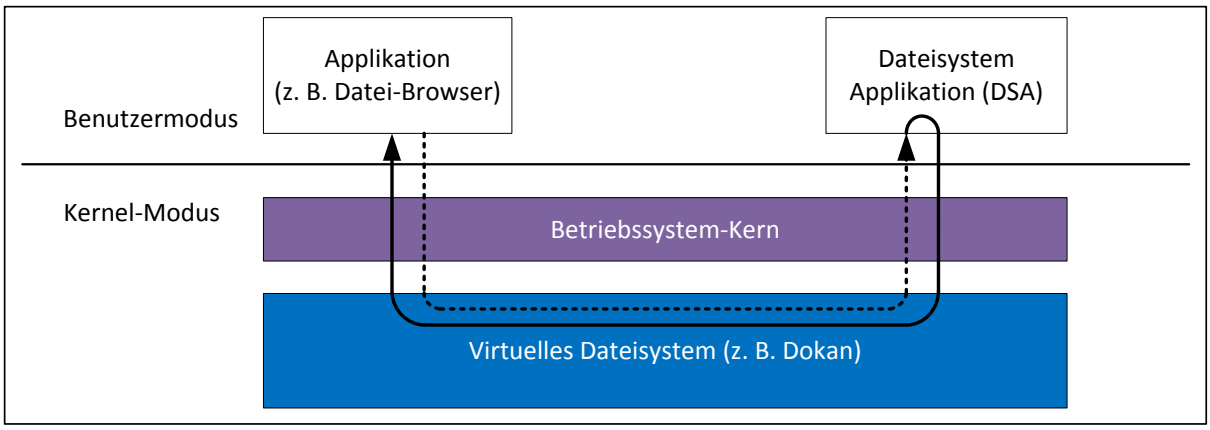

Abbildung 13: Aufruf einer Dateisystem-Applikation (DSA) aus einem Anwender-Prozess (Applikation) mittels des Dokan-Systemtreibers

2.2.4 Speicherkonzepte

Cacheverfahren

Eine Definition von Cache-Speichern — nachfolgend *Cache* genannt — gibt [31]:

> *„Cache is the name given to the highest or first level of the memory hierarchy encountered once the address leaves the processor. Since the principle of locality applies at many levels, and taking advantage of locality to improve performance is popular, the term cache is now applied whenever buffering is employed to reuse commonly occurring items. Examples include file caches, name caches, and so on.“*

Ein Cache kann potenziell immer dort eingesetzt werden, wo zwei verschiedene Systeme mit unterschiedlichen Geschwindigkeiten arbeiten und das schnellere System bei einer Anfrage auf das langsamere System warten muss. Im ungünstigsten Fall nimmt das schnellere System die Geschwindigkeit des langsameren Systems an und wird damit ausgebremst. Die Größe des Caches ist dabei limitiert, denn könnte der Cache alle Daten des vermeidlich langsamen Datenträgers speichern, würde er diesen einfach ersetzen.

In einem Dateisystem puffert ein Dateicache (File-Cache) Datenblöcke eines Datenträgers, die entweder kürzlich genutzt wurden oder mit hoher Wahrscheinlichkeit in naher Zukunft genutzt werden. Hierfür wird üblicherweise ein Teil des schnellen RAM-Speichers reserviert. Fordert ein Programm einen gewissen Datenblock vom Datenträger an und dieser Block ist bereits im Cache (sog. *Cache-Hit*), kann der Datenblock vom Cache direkt an das Programm übergeben werden — ein Zugriff auf den langsamen Speicher wird hierbei nicht benötigt. Ist der Datenblock allerdings nicht im Cache (sog. *Cache-Miss*), muss er vom externen Speicher geladen und in den Adressraum des Programms, sowie in einen freien Bereich des Caches, kopiert werden. Beim Schreiben von Daten, die nicht im Cache vorhanden sind, wird freier Speicher im Cache bereit gestellt. Die Daten werden dann in den Cache kopiert und das sog.

Dirty-Bit gesetzt. Dieses Bit gibt an, dass die Daten in diesem Block zurück auf den Datenträger geschrieben werden müssen. Dieser Vorgang wird später ausgeführt und bietet Optimierungspotenzial, wie in [31] beschrieben ist.

Eine allgemeine Beschreibung zur Verwaltung des Caches zeigt [22]:

> *„Managing a cache is primarily a matter of deciding what to keep in the cache and what to kick out of the cache when the cache is full. This management is crucial to the performance of the cache. If useful data is dropped from the cache too quickly, the cache won't perform as well as it should. If the cache doesn't drop old data from the cache when appropriate, the useful size and effectiveness of the cache are greatly reduced."*

Es existieren verschiedene Strategien, geeignete Blöcke im Cache freizugeben, wenn dieser voll ist. Sog. *Ersetzungsstrategien* (engl. *Replacement Policies*), verwerfen die Blöcke auf Grund des Zeitpunktes ihrer Einlagerung. Der bekannteste Vertreter dieser Kategorie ist *First In First Out* (FIFO). Hierbei wird der Block als erstes verworfen, der als erstes in den Cache eingefügt wurde und demnach am längsten im Cache vorhanden ist. R. Brause zeigt in [4], wie dieses Verfahren durch den sog. *Second-Chance-Algorithmus* optimiert werden kann. Eine weitere Gruppe von Ersetzungsstrategien führt Protokoll, wie oft ein Datenblock während seiner Einlagerungszeit im Cache angefragt wurde. Dazu wird bei einer Anfrage gespeichert, zu welchem Zeitpunkt der Block aufgerufen wird. Müssen Daten aus dem Cache entfernt werden, eignen sich die Datenblöcke für eine Ersetzung, die am längsten nicht mehr genutzt wurden. Ähnliches gilt für *Least Frequently Used* (LFU) — hierbei werden die Blöcke ersetzt, die am seltensten angefragt wurden.

Die Herausforderungen und auftretenden Problemstellungen bei der Verwendung verfügbarer Cache-Verfahren in der Filmproduktion und Filmdistribution werden in Abschnitt 2.3 aufgezeigt.

Redundancy Array of Independent[10] Disks (RAID)

Motiviert durch steigende Leistung von Prozessoren und der Tatsache, dass verfügbarer Speicher zwar immer größer, doch nur geringfügig schneller wird, veröffentlichten Patterson et al. [80] Konzepte, die beschreiben, wie mehrere Speichersysteme logisch so miteinander verbunden und betrieben werden können, dass die folgenden Problemstellungen adressiert werden:

- Ausfallsicherheit: Daten sollten über verschiedene Mechanismen auch dann wieder herstellbar sein, wenn ein oder mehrere Festplatten in einem Verbund ausfallen.

[10] Das damalige Akronym verwendete noch den Ausdruck „Inexpensive", heute ist jedoch „Independent" gebräuchlich.

- Durchsatzsteigerung: Daten sollten schneller auf Speichereinheiten geschrieben und von diesen Speichern lesbar sein, als das mit einzelnen Speichersystemen möglich ist.

Dabei ist es nicht zwingend nötig, dass die verwendeten Speicher identisch sind. Eine Verwendung von gleichen Speichersystemen reduziert die Komplexität allerdings deutlich. Nachfolgend sollen die für diese Arbeit relevanten Konfigurationen erläutert werden. Weitere RAID-Konfigurationen, werden unter [80,81] beschrieben.

1. RAID 0: In dieser Konfiguration werden die Daten in sog. *Stripes* unterteilt und parallel auf die verfügbaren Speichereinheiten geschrieben, respektive, von diesen Speichern gelesen. RAID 0 bietet keine Redundanz der Daten. Beim Ausfall eines Speichers sind die Daten nicht wieder herstellbar. Vorteil dieser Variante ist die Erhöhung der Schreib-/Lesegeschwindigkeit, da von allen Speichern parallel gelesen bzw. auf alle Speicher parallel geschrieben werden kann.

2. RAID 1: Hierbei werden die zu schreibenden Daten aus Gründen der Ausfallsicherheit auf mindestens zwei Speichersysteme geschrieben. Dadurch wird eine Redundanz von 100% erreicht. Zuständig für die Spiegelung beim Schreibvorgang der Daten ist der Festplatten-Controller. Lesevorgänge können von einem der beiden Speichersysteme bedient werden. Während den Schreib-/Lesepausen überprüft der Controller ständig die Integrität der Daten auf der Hauptplatte und allen Redundanzplatten. Die Schreibgeschwindigkeit wird — im Vergleich zur Verwendung nur einer Festplatte ohne Spiegelung — nicht beschleunigt.

3. RAID 3 und RAID 4: Die Konfigurationen bestehen aus *N* Speichern. Bei RAID 3 unterteilt der Controller die zu speichernden Daten in Blöcke der Größe *N-1*. Jedes Byte eines Blocks wird auf einen Speicher (Datenspeicher) kopiert. Parallel wird für den aktuellen Block ein sog. *Paritätsdatensatz* berechnet, durch den beim Ausfall eines Datenspeichers die Nutzdaten zurückberechnet werden können. Der Paritätsdatensatz wird auf die verbleibende Speichereinheit kopiert. Bei RAID 4 werden die Daten nicht byte-weise, sondern blockweise unterteilt, wodurch der Schreib- und Lesezugriff beschleunigt werden kann. Beide Konfigurationen erlauben eine parallele Ein- und Ausgabe und bieten dadurch höhere Bandbreiten, als bei der Verwendung eines einzelnen Datenspeichers. Allerdings wird die Schreibgeschwindigkeit durch die Berechnung der Paritätsdaten verringert.

4. RAID 5: Die RAID 5-Konfiguration eignet sich für unabhängige, hohe Datendurchsätze. Die Anzahl an genutzten Speichereinheiten ist nicht begrenzt. Je größer die Anzahl der Speichersysteme im RAID 5-Verbund, desto höher ist der Durchsatz bei unabhängigen Anfragen an das System. Wie bei RAID 3 und RAID 4 werden Paritätsdaten im RAID 5-Controller gebildet. Jedoch werden diese Paritätsdaten nicht auf einer dedizierten Speichereinheit gesichert, sondern abwechselnd auf allen Datenträgern

verteilt (siehe Abbildung 14). Der Vorteil dieser Variante ist, dass beim Lesen alle Platten parallel zur Verfügung stehen, da die Paritätsdaten nur im Fehlerfall benötigt werden. Fällt eine Speichereinheit aus, kann das System weiter betrieben werden — hierbei müssen die Daten der ausgefallenen Speichereinheit durch die verbliebenen Systeme und den entsprechenden Paritätsdaten errechnet werden.

Weitere RAID-Konfigurationen, wie z. B. RAID 2, RAID 6, RAID 7 und RAID 1E, sowie die Kombination von RAID-Konfigurationen sind darüber hinaus möglich [81].

Die Herausforderungen und auftretenden Problemstellungen bei der Verwendung verfügbarer RAID-Verfahren in der Filmproduktion und Filmdistribution werden in Abschnitt 2.3 aufgezeigt.

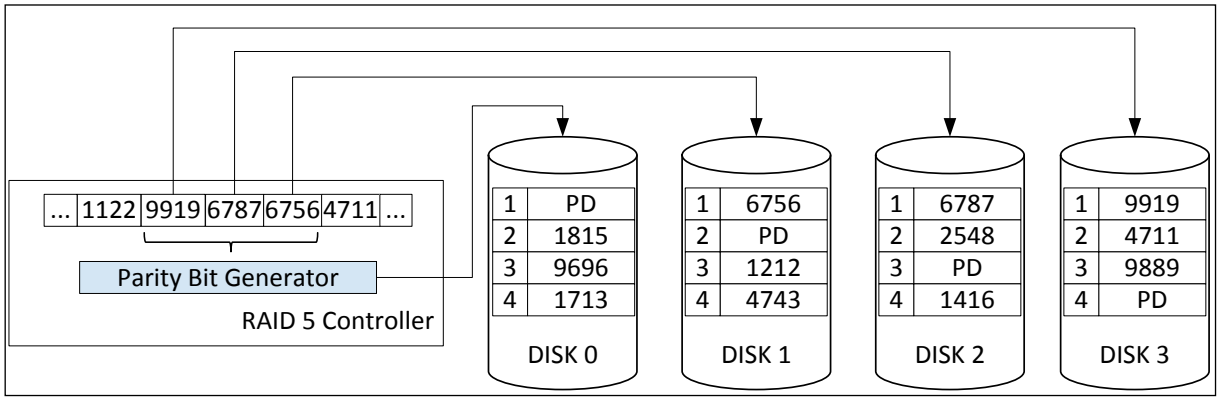

Abbildung 14: Anordnung eines RAID 5-Systems inkl. Controller zur Generierung der Paritätsdaten. Erhöhung der Effizienz, da auf alle Datenspeicher parallel zugegriffen werden kann. Für das Auslesen sind drei parallele Zugriffe (statt vier wie bei einer RAID 4 Konfiguration) nötig: (1) Disk0-1 + Disk1-1 + Disk2-1 + Disk3-2, (2) Disk0-2 + Disk1-2 + Disk2-3 + Disk3-3, (3) Disk0-3 + Disk1-4 + Disk2-4 + Disk3-4

2.3 Herausforderungen bei der heutigen Verarbeitung von Mediendaten

Der Datentransfer und die Verarbeitung von Multimediadaten stellt eine Vielzahl von Herausforderungen innerhalb von Computersystemen dar, die u. U. durch Nutzung der Skalierbarkeit moderner Kompressionsverfahren gelöst werden können. Die folgende Auflistung zeigt die in dieser Arbeit identifizierten und adressierten Problemstellungen:

1. Während der (Post-) Produktion von Filmen werden täglich Testsequenzen (sog. *Dailies*) vom Filmset zur Sichtung an verschiedene Empfänger versendet. Je nach Rolle des Empfängers im aktuellen Projekt ist es dabei wünschenswert, verschiedene Varianten der erstellten Dailies mit unterschiedlichen Qualitäten und Auflösungen bereit zu stellen. Die in Abschnitt 2.2.3 vorgestellten Dateisysteme besitzen keine Kenntnis über die Struktur der Daten, die sie speichern. Eine effektive Nutzung der Skalierbarkeit von Mediendaten kann somit nicht erreicht werden. Weder FAT, NTFS noch die

Familie der Extended-Dateisysteme erlauben die Vergabe von Zugriffsrechten auf Teile einer Datei. Rechtevorgaben, bei denen bestimmte Nutzer die volle Auflösung/Qualität einer Videodatei sehen können, wohingegen andere Nutzer lediglich Zugriff auf eine reduzierte Variante erhalten, können nur durch Erstellung von sog. *Proxy-Varianten* erreicht werden. Die Erstellung und Speicherung bindet dabei wichtige Hard- und Softwareressourcen. Hinzu erschwert sich die Verwaltung der einzelnen Dateien, da Änderungen nicht automatisch in allen Proxy-Varianten durchgeführt werden.

Ausgehend von den Limitierungen heutiger Dateisysteme werden in den Abschnitten 4.1 und 5.2 neue Konzepte zur Verwaltung skalierbarer Medien vorgestellt.

2. Beim Beschreiben von Datenspeichern mit beweglichen Teilen (Festplatten) oder Datenspeicher mit sequenziellem Zugriffsverhalten (Bandspeichern) wird die Skalierbarkeit von Mediendaten heute nicht berücksichtigt. Es kommt u. a. zu Situationen, bei denen die Leistung der Speichersysteme suboptimal ist: (1) Die Schreib-/Leseköpfe einer konventionellen Festplatte müssen bei Abruf einer reduzierten Variante mehrfach neu positioniert werden, auch wenn die Daten beim Schreiben auf den Datenträger sequenziell abgelegt wurden. Besonders das Überspringen von nicht benötigten Datenpaketen durch Auslassen einer Skalierungsstufe ist bei sequenziellem Speichern problematisch und kann zu starken Leistungseinbrüchen führen. Um die volle Leistung einer Festplatte ausnutzen zu können, müssten die Daten in diesem Fall vor der Nutzung umkopiert werden. Dieser Kopiervorgang ist zeitaufwendig und verlangt ein gewisses Maß an Zusatzverwaltung zur Sicherstellung der Datenintegrität zwischen den verschiedenen Versionen.

   Ausgehend davon, dass solche Zugriffsverhalten für bestimmte Branchen bzw. Projekte vorab bekannt sind, könnten die Daten bereits im Vorfeld derart gespeichert werden, dass die Festplattenleistung für den entsprechenden Anwendungsfall optimal ist. Auch wenn die klassische Festplatte bei Desktop-Rechnern zunehmend von flashbasierten Speichern verdrängt wird, ist eine Verwendung in der Filmpostproduktion und Distribution noch über Jahre zu erwarten. Ausschlaggebend sind hier vor allem die geringeren Kosten im Vergleich zu SSD-Festplatten. (2) Bei Speichersystemen mit sequenziellem Zugriffsverhalten kann die Datenmenge pro Bild größer sein, als der Datenträger später beim Auslesen der Datei unter Echtzeitbedingung (vgl. 1) liefern kann. Daten von Bandspeichern müssen deshalb erst auf ein schnelleres Medium umkopiert werden, bevor sie genutzt werden können – diese Kopierprozesse sind zeitaufwendig. Die Verwendung von Bandspeichern ist besonders in der Domäne der digitalen Archivierung von Filmmaterial noch über mehrere Jahrzehnte zu erwarten.

   Flashbasierte Speicher sind in der Lage, Daten für die Bereiche Digitales Kino und Digitale Archivierung in Echtzeit zu liefern. Eine Herausforderung liegt hier in der

Minimierung, des durch SSDs verbrauchten Speicherplatz, da Flashspeicher — im Vergleich zu Magnetspeicher — teuer ist. Neuere Betriebssysteme speichern deshalb häufig angefragte Systemdaten auf einem Flashspeicher und weniger häufig angefragte Daten auf einem kostengünstigen Magnetspeicher mit wahlfreiem Zugriffsverhalten. Eine solche Priorisierung der Datenpakete ist ebenfalls für skalierbare Medien denkbar.

Ausgehend von den Limitierungen heutiger Dateiorganisationen in Consumer-Dateisystemen werden in Abschnitt 4.2 speicheradaptive Verfahren mit besonderem Fokus auf skalierbare Medien vorgestellt.

3. Die in Abschnitt 2.2.4 vorgestellten RAID-Systeme teilen Daten ebenfalls blockbasiert und ohne Berücksichtigung der internen Struktur der Daten auf, verteilen sie dann aus verschiedenen Motivationen — wie z. B. Steigerung der Durchsatzgeschwindigkeit oder Datenredundanz — auf mehreren Datenträgern. Hierdurch werden auch die Datenpakete skalierbarer Medien willkürlich getrennt, wodurch beim Lesen der Daten erneut alle Datenträger zur Verfügung stehen müssen. Somit entstehen Datenstränge skalierbarer Medien, die nur durch Zusammenschluss aller verwendeten Datenträger einen validen Datenstrom ergeben.

   Dieser Umstand kann durch eine für skalierbare Medien angepasste Verteilungsstrategie behoben werden, die in Abschnitt 4.2.2 vorgestellt wird.

4. Für Videoausspielung ist die Echtzeitfähigkeit aller verwendeten Komponenten — also Datenspeicher, die Schnittstelle zwischen Datenspeicher und dem abspielenden Computersystem sowie der verwendete Prozessor — eine wichtige Voraussetzung, damit entsprechende Filme ohne Bildaussetzer abgespielt werden können. Hierbei stellen die verwendeten Schnittstellen häufig einen Engpass dar, besonders bei sehr hohen Datenraten, wie sie in der professionellen Filmproduktion verwendet werden. Ein Vergleich der in Abschnitt 2.2.1 vorgestellten Computerschnittstellen mit den in Tabelle 3 angegebenen Datenraten zeigt, dass besonders die älteren Versionen von USB und FireWire nicht in der Lage sind, die Daten für Standardanwendungen im Bereich Digitales Kino und/oder der digitalen Archivierung entsprechend schnell genug zu liefern. Ein ruckelndes Abspielen auf Grund von Bildaussetzern ist die Folge. Dabei sind besonders diese Technologien weit im Markt verbreitet.

   Zur Umgehung der auftretenden Engpässe werden heute üblicherweise die entsprechenden Daten vor einer Echtzeitausspielung auf einen Datenträger umkopiert, der über eine leistungsstarke Schnittstelle angebunden ist, um die Daten schnell genug liefern zu können. Dieser sog. *Umkopierungsprozess* ist zeitintensiv, führt zu einer redundanten Speicherung der Filmsequenz und bindet somit zusätzliche Speicherressourcen.

Neuere Schnittstellen erreichen höhere Datenraten und sind somit schnell genug für heutige Standardpakete. Allerdings wird sich zukünftig auch die Datenrate und Bildwiederholfrequenz derart erhöhen, dass selbst Thunderbolt (vgl. Tabelle 4) unter Verwendung mehrerer Verbindungen (Lanes) an die Datentransfergrenzen stoßen wird.

Ausgehend von den Limitierungen heutiger Schnittstellen werden in Abschnitt 5.1 echtzeitfähige Einlese-Strategien unter Ausnutzung skalierbarer Medien vorgestellt.

5. Einlagerungs- und Ersetzungsstrategien von den in Abschnitt 2.2.4 vorgestellten Cache-Verfahren arbeiten blockbasiert. Muss Speicher freigegeben werden, fällt die Entscheidung entsprechend an Blockgrenzen und nicht auf Grund von Paketgrenzen innerhalb skalierbarer Medien. Diese Eigenschaft gilt sowohl für Cache-Einlagerungsverfahren, wie auch Cache-Ersetzungsstrategien. Hierdurch werden Blöcke verworfen, die u. U. Grundinformationen der Bilder, Videos oder Musikstücke beinhalten. Diese Blöcke müssen erneut vom Massenspeicher gelesen werden, wenn ein Nutzer die Datei erneut anfragt. Hierdurch kann es zu Aussetzern bei der Wiedergabe kommen.

   Ausgehend von den Limitierungen dieser Cache-Verfahren in Consumer-Betriebssystemen (Linux, Mac OS, Windows) werden in Abschnitt 5.3.2 Cache-Verfahren mit besonderem Fokus auf skalierbare Medien vorgestellt.

3. Substitutionsmethode

Bevor die konkreten Algorithmen zur Verbesserung von Arbeitsabläufen gezeigt werden, wird in diesem Kapitel ein neues Verfahren zur Kompensation fehlender Bereiche skalierbarer Multimediadaten vorgestellt, das im Rahmen dieser Arbeit entwickelt wurde.

3.1 Motivation

Ausschlaggebend ist die Erkenntnis, dass bei der Anfrage von Multimediadaten nicht immer gewährleistet ist, dass diese unter bestimmten Randbedingungen rechtzeitig bereitgestellt werden können. Die Gründe dafür sind — ebenso wie die Randbedingungen — sehr vielfältig. Im Fokus dieser Arbeit geht es vorzugsweise um die Anlieferung großer Multimediadaten unter Echtzeitanforderung, wie es bspw. beim Abspielen eines Videos der Fall ist. Limitierende Faktoren sind u. a. Durchsatzraten der verwendeten Datenträger und/oder Schnittstellen zwischen Datenträgern und Computer-Systemen für die Wiedergabe. Es stellt sich demnach die Frage, ob fehlende Daten ggf. ersetzt werden können, ohne die vorhandenen Bildinformationen zu verschlechtern bzw. zu verfälschen.

3.2 Vorstellung der Substitutionsmethode

Zur Erläuterung der Substitutionsmethode wird ein typischer Anwendungsfall bei der Wiedergabe von Einzelbildsequenzen betrachtet. Z. B. initiiert ein Wiedergabeprozess eine Leseanfrage auf eine Bildsequenz mit einer Bildwiederhohlrate von 24 fps (engl. *frames per second*) an das Dateisystem. Für jedes Einzelbild erzeugt der Prozess ein sog. *Datei-Handle*, wodurch er Zugriff auf die Datei erhält. Über das Datei-Handle werden anschließend die Bilddaten in den Prozess-Kontext kopiert, üblicherweise decodiert und zur Weiterverarbeitung bereitgestellt. Eine Möglichkeit der Weiterverarbeitung stellt dabei die Visualisierung über Grafikkarte und Monitor dar.

Idealerweise werden dem Wiedergabeprozess die gespeicherten Einzelbildsequenzen schnell genug vom Dateisystem zur Verfügung gestellt. In diesem Fall können die vorgeschalteten Komponenten, wie Datenspeicher und Datenschnittstellen, die Multimediadaten schnell genug liefern. Allerdings kann diese Eigenschaft nicht immer gewährleistet werden, da verschiedene Komponenten in der Verarbeitungskette nicht zwangsläufig leistungsstark genug sind. Beispielhaft kann die genutzte Festplatte der limitierende Faktor sein, wodurch dem Wiedergabeprozess eine angefragte Datei nicht innerhalb der Zeit angeliefert wird, die ihm insgesamt für die Decodierung und Anzeige zur Verfügung steht. Üblicherweise reagieren heutige Abspielsysteme derart, dass das aktuelle Bild nicht decodiert und demnach

auch nicht angezeigt wird. Die Grafikkarte zeigt weiterhin das zuletzt erfolgreich decodierte Bild an. Hierdurch kommt es zu Bildaussetzern bzw. Standbildern, die ein Betrachter als Ruckeln wahrnimmt.

Dabei ist zu beachten, dass die Festplatte u. U. zwar nicht schnell genug ist, um das komplette Bild bzw. die komplette Bildsequenz an den Wiedergabeprozess zu liefern, jedoch ist es denkbar, dass größere Teile der Multimediadaten übertragen werden können. Ist eine fehlerfreie Decodierung von komprimierten, nicht-skalierbaren Bildern üblicherweise nicht möglich, kann es bei einer skalierbaren Variante durchaus der Fall sein, dass die von der Festplatte gelieferten Daten bereits eine Subvariante der originalen Bilddatei beinhalten. Anstatt also ein Bild überhaupt nicht weiter zu verarbeiten, könnte der Wiedergabeprozess bei Verwendung skalierbarer Medien eine Subvariante anzeigen.

Damit der aufrufende Wiedergabeprozess in diesem Beispiel eine vollständige, valide Datei erhält, füllt die Substitutionsmethode die fehlenden Daten der skalierbaren Datei durch Pseudodaten auf und erzeugt dadurch eine virtuelle Datei, die in ihrer internen Struktur und der Dateigröße der originalen Datei auf der Festplatte entspricht. Zur Generierung der Pseudodaten werden die bereits von der Festplatte eingelesenen Daten verwendet, da diese u. a. Auskunft über die interne Struktur der skalierbaren Datei geben. Dabei werden die fehlenden Daten durch eine Serie von Werten ersetzt, die dem nachgeschaltetem Decoder anzeigen, dass keine weiteren Nutzdaten mehr zur Verfügung stehen. Selbstverständlich darf keine Veränderung des bereits von der Festplatte gelesenen Teilbildes durch die Pseudodaten erfolgen.

Für die sinnvolle Anwendung der Substitutionsmethode innerhalb der noch folgenden Lösungsansätze dieser Arbeit, sind die folgenden Eigenschaften elementar:

1. Die Anwendung erfolgt auf bereits encodierten Mediendaten. Es ist somit nicht nötig, vorhandene Dateien zusätzlich zu decodieren, um fehlende Daten mit Pseudodaten zu ersetzen. Hierzu ist es allerdings nötig, dass die bereits gelesenen Metadaten ausreichende Informationen über die interne Struktur der skalierbaren Datei beinhalten.

2. Die Ausführung der Substitutionsmethode erfolgt ausschließlich im Arbeitsspeicher — in Abhängigkeit der Anwendung wahlweise auf dem Wiedergabegerät oder auf dem Dateiserver. Einfache Durchsatzmessungen haben gezeigt, dass ein handelsüblicher Zweikernprozessor (z. B. 2.3 GHz Dual-Core Intel Core 5 mit 3 MB On-Chip L3-Cache) viele hundert Bilder pro Sekunde mit der Substitutionsmethode bearbeiten kann, wenn die Daten schnell genug von den verwendeten Datenträgern angeliefert werden bzw. nach Anwendung der Substitutionsmethode schnell genug vom Nachfolgeprozess entgegen genommen werden.

3. Die Dateistruktur bleibt identisch. D. h. Dateigröße und interne Struktur der originalen Datei werden übernommen. Für eine aufrufende Applikation ist die Ausführung der

Substitutionsmethode somit transparent. Hierdurch ist keine zusätzliche Client/Server-Architektur nötig, um das Fehlen von Nutzdaten zu signalisieren.

Anwendung findet die Substitutionsmethode immer dann, wenn die encodierten Daten nicht schnell genug geliefert werden können bzw. nicht länger zur Verfügung stehen. Dies ist z. B. der Fall, wenn eine Festplatte im Dateiserver nicht den entsprechenden Datendurchsatz erreicht, um eine Bildsequenz in Echtzeit zu liefern oder eine zwischengeschaltete Schnittstelle zu langsam ist und dadurch einen Engpass bildet.

3.3 Anwendung bei JPEG 2000

[11]Die Anatomie sog. *Code-Block-Pakete* innerhalb des Codestreams (vgl. Abbildung 15) erlaubt es, leere Einheiten einzubauen (siehe auch [113]). Dabei gibt das erste Bit des Paket-Headers an, ob das Paket leer ist oder tatsächlich Daten enthält. Ist das erste Bit mit dem Wert 0 initialisiert, besteht das komplette Paket nur aus einem Byte, bei dem die übrigen sieben Bits einen undefinierten Zustand aufweisen und nicht weiter ausgewertet werden. Enthält das erste Bit des Paket-Headers den Wert 1, folgen die Paketgröße und andere Metadaten, die zum Decodieren benötigt werden. Hierbei ist zu bemerken, dass das Paket noch immer leer sein kann, also keine Nutzdaten im Paketrumpf vorhanden sind. Wie in Abbildung 15 zu erkennen ist, werden alle Paket-Symbole (engl. *Tags*) einzelner Code-Blöcke zu einem gesamten Paket-Header zusammengefasst, bevor die Nutzdaten aller Code-Blöcke folgen. Die Reihenfolge bleibt dabei in beiden Bereichen identisch. Für die Anwendung der Substitutionsmethode ist entscheidend, dass leere Einheiten in den Codestream eingebaut werden können. Für eine praktikable Anwendung muss es jedoch möglich sein, zumindest grob innerhalb eines Codestreams navigieren zu können. Hierzu wird jedoch eine grobkörnigere Struktur in JPEG 2000 verwendet, die sich auf die *logischen Pakete* bezieht (vgl. auch 2.1.2). Ein logisches Paket kann viele Code-Block-Pakete enthalten, die Code-Block-Pakete können leer sein. In welchem logischen Paket später entsprechende Code-Block-Pakete angeordnet werden, wird bei der Codierung durch die Progressionsreihenfolge festgelegt. Für die grobkörnige Navigation im Codestream ist es nötig, dass die Längen aller logischen Pakete im Header der JPEG 2000-Datei vorhanden sind. Dies ist durch die sog. *Paketlängen-Marker* möglich.

Abbildung 16 zeigt die Anordnung des Main-Headers einer JPEG 2000-Datei, mehrerer Kacheln (engl. *Tiles*) und den entsprechenden Detailaufbau eines Tile-Headers, der von den sog. *Paketdaten* gefolgt wird. Hierbei handelt es sich um die Code-Block-Pakete, die zuvor encodiert wurden und bereits in logische Pakete zusammengefasst sind. Weiter erlaubt der Tile-Header die Vergabe der Paketlängen-Marker, welche die Länge der logischen

[11] Teilaspekte dieses Abschnitts wurden in [105] sowie [108] veröffentlicht.

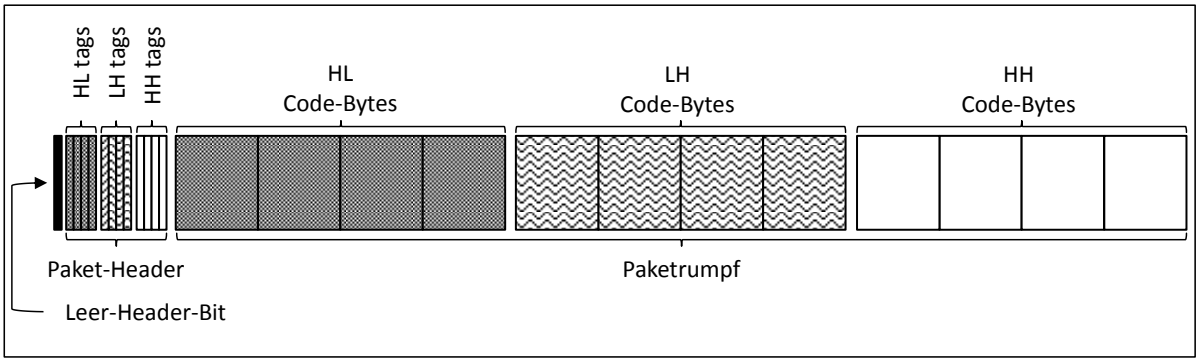

Abbildung 15: Code-Block-Paketstruktur innerhalb eines JPEG 2000-Codestreams nach [113]

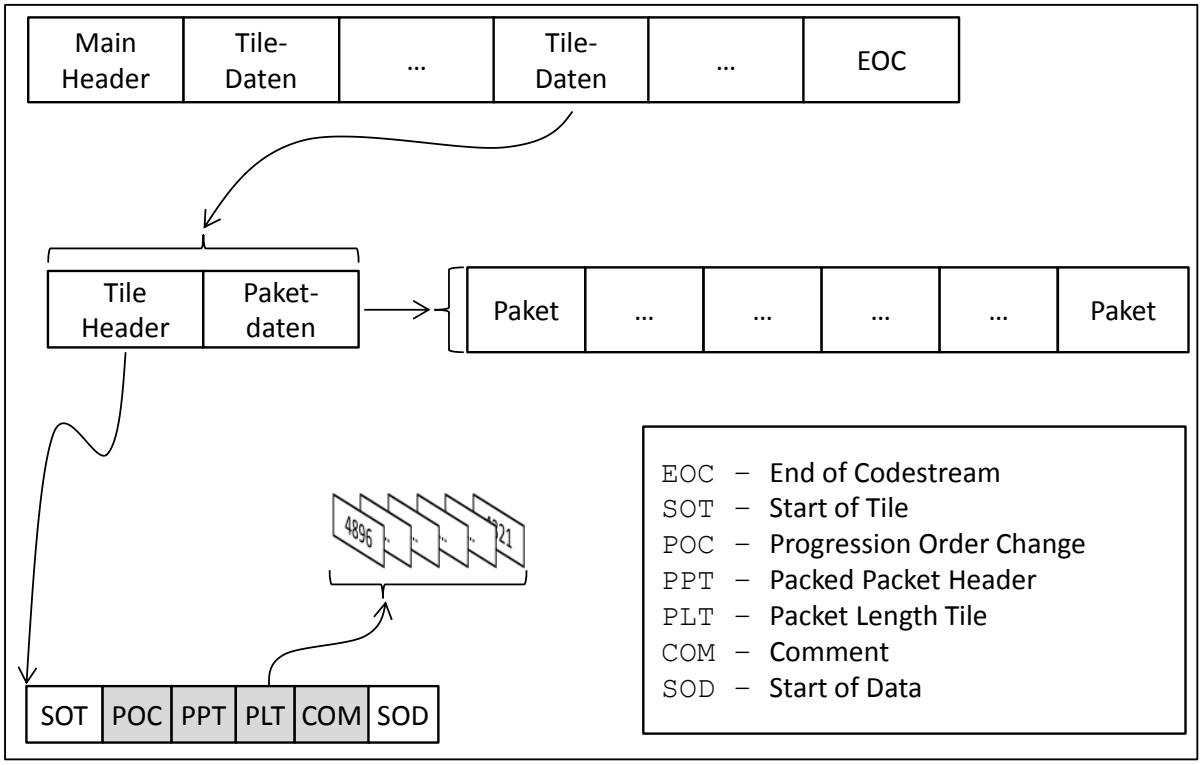

Abbildung 16: Detailaufbau eines JPEG 2000-Codestreams mit mehreren Tiles und PLT-Marker nach [113]

JPEG 2000-Pakete beinhalten. Dieses sog *Packet Length Tile-part(PLT)-Marker*-Segment ist dabei optional, wird jedoch essentiell für die Substitutionsmethode benötigt, da es hierdurch möglich wird, logische Pakete innerhalb eines Codestreams direkt anzuspringen, ohne vorher durch die einzelnen Code-Block-Pakete navigieren zu müssen.

Die nachfolgende Erklärung der Substitutionsmethode soll anhand eines einfach aufgebauten Codestreams mit einer Tile-Einheit erläutert werden — ein Aufbau, der im Bereich Digitales Kino und für die digitale Archivierung verwendet wird. Der Main- sowie der Tile-Part-Header werden als erste Elemente in einer JPEG 2000-komprimierten Datei gespeichert und umfassen üblicherweise nur wenige hundert Bytes. Demnach kann davon ausgegangen werden, dass die Header-Daten nach einem ersten Zugriff auf die Datei zur Verfügung stehen, auch wenn diese nicht komplett vom Datenträger gelesen werden kann. In Abbildung 16 wird

deutlich, dass es anhand dieser Metainformationen möglich wird, eine virtuelle Datei — bisher noch ohne reale Nutzdaten — nachzubauen, da die Längen, bzw. die Offsets der einzelnen logischen Pakete, mit der sich die Längen berechnen lassen, im Tile-Header abgespeichert sind. Diese virtuelle Datei kann durch weitere Leseanfragen mit realen Daten gefüllt werden. Entsprechend dem in Abschnitt 3.2 skizzierten Anwendungsfall zeigt Abbildung 17-a die Anwendung der Substitutionsmethode unter Verwendung einer JPEG 2000-Datei. Hierbei sind einige logische Pakete innerhalb der virtuellen Datei mit gültigen Daten belegt — andere Bereiche noch nicht initialisiert. In Abbildung 17-a ist dies durch einen originalen Codestream und drei virtuelle Versionen exemplarisch gezeigt. Dabei beinhaltet der originale Codestream einen Header *H*, der Main-Header und Tile-Header zusammenfasst. Innerhalb des Tile-Headers wurden während der Codierung entsprechende PLT-Marker-Segmente abgespeichert, so dass die Länge der logischen Pakete bekannt ist und damit eine virtuelle Variante der Datei nachgebaut werden kann, sobald *H* eingelesen wurde. Weiter zeigt Abbildung 17-a, dass bei den drei Versionen eine unterschiedliche Anzahl von logischen Paketen von der originalen Datei eingelesen wurde. Pakete, die nicht oder nur teilweise eingelesen werden konnten, werden durch die Substitutionsmethode mit einer entsprechenden Serie von Nullwerten ersetzt. Abschließend wird das letzte Byte mit dem Codestream-Ende-Marker belegt und an den aufrufenden Prozess übergeben. Durch die Anatomie der Code-Block-Pakete wird die Serie von Nullpaketen während der Decodierung als eine Reihe von leeren Code-Block-Paketen interpretiert. Beim ersten substituierten, logischen Paket angekommen, interpretiert der Decoder nur noch leere Code-Block-Pakete — entsprechend der Anatomie in Abbildung 16 — bis der anschließende Codestream-Ende-Marker eingelesen wird und die nachfolgenden Decodierungsstufen aufgerufen werden.

In Abhängigkeit der ersetzten Skalierungsstufe wirkt sich die Substitutionsmethode unterschiedlich aus. Wird ein logisches Paket für die nächsthöhere Auflösungsstufe durch die Substitutionsmethode mit Nullwerten ersetzt, erhalten entsprechend alle Wavelet-Koeffizienten in den zugehörigen Subbändern den Wert Null. Die nachfolgende Wavelet-Stufe nutzt diese Koeffizienten im Synthese-Schritt, wodurch das Endergebnis, im Vergleich zum nicht-substituierten Datenstrom, weniger bzw. keine Details aus den Hochpass-Bändern beinhaltet. Die räumliche Bildgröße allerdings wird durch die Substitution nicht verändert.

Kommt es zur Ersetzung eines logischen Paketes für die nächsthöhere Qualitätsschicht, führt die Substitutionsmethode zur Präzisionsreduktion der Koeffizienten, die durch die Bitplanes abgebildet werden. Durch die Ersetzung von Nullwerten werden die Bitplanes für die niederwertigen Bits (vgl. Abbildung 5) abgeschnitten — entsprechend verringert sich die Bildqualität im Vergleich zum nicht-substituierten Datenstrom nach Durchführung der anschließenden Decodierungsschritte.

Die Ersetzung eines Paketes für eine (Farb-)Komponente führt zum Verlust dieser Komponente ab der entsprechenden Skalierungsstufe. Die Ersetzung von Positionspaketen

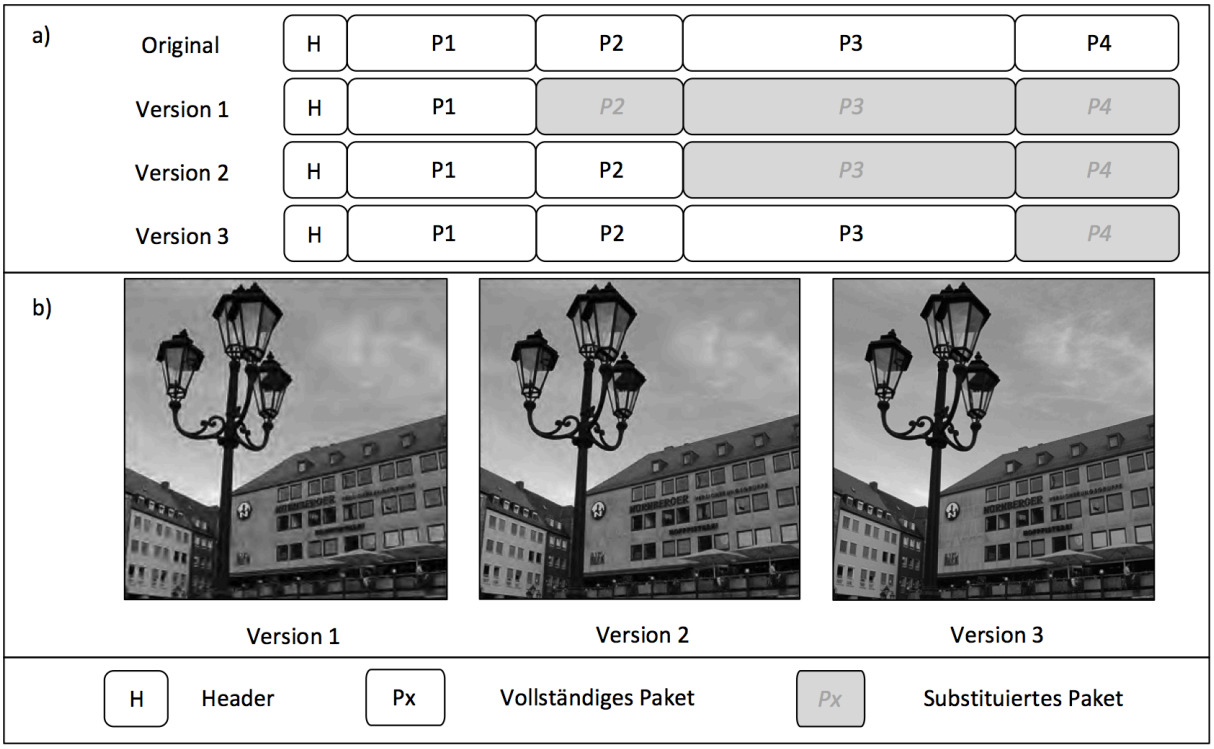

Abbildung 17: Substitutionsmethode am Beispiel einer stark vereinfachten JPEG 2000-Dateistruktur. a) Originale Datei mit verschiedenen virtuellen Versionen mit identischer Struktur und substituierten Paketen. b) Qualitätsergebnis nach Decodierung der Versionen

führt dazu, dass keine weiteren Detaildaten für den gewünschten Bereich zur Verfügung stehen.

Abbildung 17-b zeigt die decodierten Bilder der Codestream-Versionen 1-3. Wie zu erkennen ist, steigt die visuelle Qualität, wenn weniger logische Pakete durch die Substitutionsmethode mit sog. *Nullpaketen* ersetzt wurden. Zusätzlich sei noch vermerkt, dass ein Codestream, bei dem alle Datenpakete durch Null-Pakete ersetzt wurden, bereits ein valides JPEG 2000-Bild repräsentiert. Die Decodierung der Datei würde in einem grauen Bild resultieren.

4. Konzepte zur Verwaltung skalierbarer Mediendaten

In diesem Kapitel werden zwei Bereiche untersucht. Zunächst erfolgt die Entwicklung und Evaluation eines dedizierten Rechtemanagements für skalierbare Medien. Anschließend werden speicheradaptive Dateiorganisationen zur Durchsatzsteigerung von Festplatten mit rotierenden Scheiben sowie spezielle RAID-Verfahren für diverse Speichertechnologien vorgestellt.

4.1 Erweitertes Rechtemanagement

[12]Populäre Dateisysteme im Consumer-Bereich besitzen z. T. ein feingranulares Rechtemanagement, das es erlaubt, unterschiedlichen Nutzern bzw. Nutzergruppen Zugriffsrechte wie Schreiben, Lesen usw. auf Ordner und Dateien zuzuweisen. Für nicht skalierbare Medien reichen diese Konzepte weitestgehend aus, da bestimmte Zugriffsarten auf Teile einer Datei wenig sinnvoll sind. Bei der Verwendung skalierbarer (Medien-)Daten hingegen ist eine Rechtevergabe auf Teile einer Datei wünschenswert, da unterschiedliche Benutzer hierdurch Zugriff auf verschiedene Versionen einer Datei bekommen können. Mit dem Rechtemanagement heutiger Dateisysteme lassen sich Zugriffe auf unterschiedliche Versionen nur bewerkstelligen, indem entsprechend viele Proxy-Varianten berechnet werden und Anwender entsprechende Rechte für eine dieser Proxy-Varianten bekommen. Dieses Vorgehen ist im Bezug auf Rechenressourcen für die Erstellung der Proxies sowie dem zusätzlich benötigten Speicherplatz nicht optimal. Weiter erschwert die nötige Synchronisation der Proxydateien den täglichen Arbeitsprozess. Dieser Zusatzaufwand tritt ein, wenn Änderungen an nur einer Proxy-Datei vorgenommen werden, da diese Änderungen zwangsläufig in allen Proxy-Varianten nachgezogen werden müssen, um einen einheitlichen Versionsstand zu erhalten. Zwar lässt sich das Problem der Synchronisierung bereits heute — durch Nutzung skalierbarer Medien — umgehen, allerdings können verschiedene Benutzerrechte nicht mehr abgebildet werden, da ein Zugriff auf eine skalierbare Datei nicht ausschließt, dass ein Benutzer auch höhere Qualitäts- und Auflösungsstufen anfragt.

[12] Teilaspekte dieses Abschnitts wurden in [105] sowie [109] veröffentlicht.

4.1.1 Virtuelles Dateisystem mit Rechteverwaltung für skalierbare Medien

Ausgehend von den Anforderungen zur Vermeidung der Proxy-Generierung, sowie der möglichen Vergabe von Zugriffsrechten auf skalierbare Mediendateien, wurde im Rahmen dieser Arbeit ein Dateisystem entwickelt, das beide Anforderungen erfüllt. Abbildung 18 zeigt dabei das Grundprinzip des Dateisystems, bei dem reale Dateien von einer lokalen Festplatte mittels einer Zwischenschicht (Dateisystem für skalierbare Medien) zum Teil in virtuelle Varianten umgewandelt werden.

Hierbei ist eine Bildfolge skalierbarer JPEG 2000-Bilder auf der lokalen Festplatte (*Ordner C:/Sequenz1*) gespeichert. Die originalen Bilder verfügen über eine maximale räumliche Auflösung von 4096x2160 Pixeln (4K), die Progressionsreihenfolge ist mit LRCP qualitätsorientiert. Innerhalb des Dateisystems für skalierbare Medien wurden Zugriffsrechte vergeben und in einer separaten Datenbank gespeichert. Nach Abbildung 18 besitzt Benutzer A entsprechende Rechte, um auf die volle 4K-Auflösung der Bilder zugreifen zu können. Benutzer B hingegen darf lediglich auf die 2K-Versionen mit 2048x1080 Pixeln der skalierbaren Dateien zugreifen. Dabei greifen die Benutzer nicht direkt auf den realen Speicherort *C:/Sequenz1/* zu — hierzu fehlen ihnen die Zugriffsrechte. Vielmehr verfügen die Benutzer über ein Netzlaufwerk, das ihnen Zugriff auf das virtuelle Dateisystem gewährt. Fragt Benutzer A ein Bild an, kann das Dateisystem für skalierbare Medien anhand der Benutzerrechte erkennen, dass die originale Datei nicht verändert werden muss. Demnach liest die Zwischenschicht die originale Datei und liefert diese an Benutzer A zurück. Fragt Benutzer B hingegen ein Bild an, erkennt die Zwischenschicht anhand der hinterlegten Zugriffsrechte, dass die originale Version verändert werden muss, bevor diese an den aufrufenden Prozess zurückgegeben werden kann. Die Zwischenschicht ist für die Erstellung einer virtuellen Datei zuständig, die dann an Benutzer B zurückgegeben wird. Beiden Benutzern ist dabei nicht bekannt, ob sie bereits die volle Auflösung und/oder Qualität der Bilder erhalten — besonders Benutzer B werden lediglich virtuelle Dateien angeboten.

Natürlich lassen sich neben der Bildauflösung auch weitere Skalierungsoptionen verwenden. Neben der Virtualisierung von lokalen Datenträgern können auch weitere Komponenten im Netzwerk verwaltet werden, wie z. B. Netzwerkspeicher (*Network Attached Storage* – NAS) im Intranet sowie bspw. FTP-Server im Internet.

Architektur des Dateisystems für skalierbare Medien:

Abbildung 19 zeigt die realisierte Architektur des Dateisystems. Neben dem realen Datenträger, der die skalierbaren Mediendaten tatsächlich speichert, finden sich noch folgende Komponenten in der Architektur:

1. Dateisystemanwendung (DSA): Bildet die Hauptkomponente des virtuellen Dateisystems. Sie empfängt auf der einen Seite alle Anfragen der Benutzer in Form von Datei-

operationen (*stat/open/read* usw.) und kommuniziert auf der anderen Seite mit dem realen Dateisystem. Unter Zuhilfenahme der Media-Repackaging-Komponente und der Datenbank für Zugriffsrechte ermöglicht die DSA die gewünschte Funktionalität der feingranularen Rechtevergabe auf skalierbare Mediendateien.

2. Datenbank für die Zugriffsrechte: Die Datenbank speichert Informationen zum Erstellen von Zugriffskontrolllisten (engl. *Access Control Lists* – ACL), die für eine Vergabe von Zugriffsrechten auf Dateien und Ordner auf dem realen Dateisystem verwendet werden. Diese ACLs erlauben es jedoch nicht, bestimmte Bereiche einer Datei freizugeben und andere Bereiche zu sperren, wie das für den gewünschten Anwendungsfall nötig wäre. Aus diesem Grund werden zusätzliche Zugriffsrechte für den Zugriff auf Auflösungs- und Qualitätsvarianten der skalierbaren Mediendaten in der Datenbank gespeichert. Für die Grundfunktionalität des Systems müssen mindestens die folgenden Informationen in der Datenbank vorhanden sein:

    a. Benutzername und Domäne,

    b. maximale Auflösung des Videos/der Bildsequenz in Prozent, die ein Benutzer erhalten darf,

    c. maximale Qualität der skalierbaren Mediendatei in Prozent, die ein Benutzer erhalten darf,

    d. Dateipfad zur Bildsequenz auf dem realen Datenträger.

3. Media-Repackaging-Komponente (MRK): Realisiert die Neusortierung der skalierbaren Dateien. Die MRK kommt immer dann zum Einsatz, wenn dem Benutzer lediglich Zugriff auf eine reduzierte Variante der realen Datei gewährt wird. In diesem Fall muss die MRK zwei Operationen ausführen:

    a. Die Datenpakete der skalierbaren Datei müssen neu zusammengestellt werden, so dass die virtuelle Datei den Benutzerrechten des anfragenden Prozesses entspricht.

    b. Der Header der neuen, virtuellen Datei muss entsprechend der erfolgten Neusortierung angepasst werden.

    Darüber hinaus ist die MRK noch in der Lage, Metainformationen (z. B. Dateigröße) zu skalierbaren Dateien zu liefern, die in Abhängigkeit von der realen Datei und den Benutzerrechten bestimmt werden.

Weiter zeigt Abbildung 19 noch die für den Benutzer sichtbaren Komponenten wie den virtuellen Datenträger, der in Form eines Laufwerksbuchstabens sichtbar ist, sowie den Wiedergabeprozess. Dies kann sowohl ein Datei-Browser wie auch eine Abspielsoftware für Videodaten sein. Die Komponenten Betriebssystem und Dateisystemtreiber arbeiten dabei im

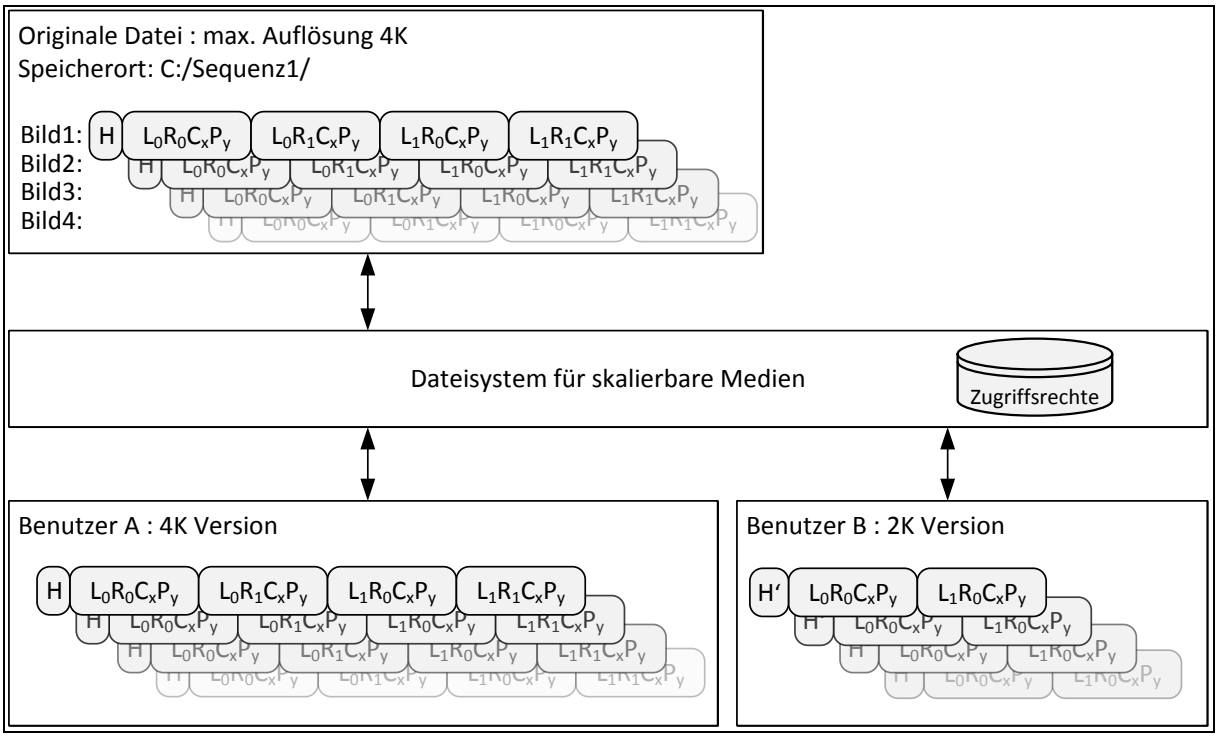

Abbildung 18: Grundprinzip der Rechtevergabe auf skalierbare Medien am Beispiel einer Einzelbildsequenz im JPEG 2000-Format, Detailbeschreibung im Fließtext

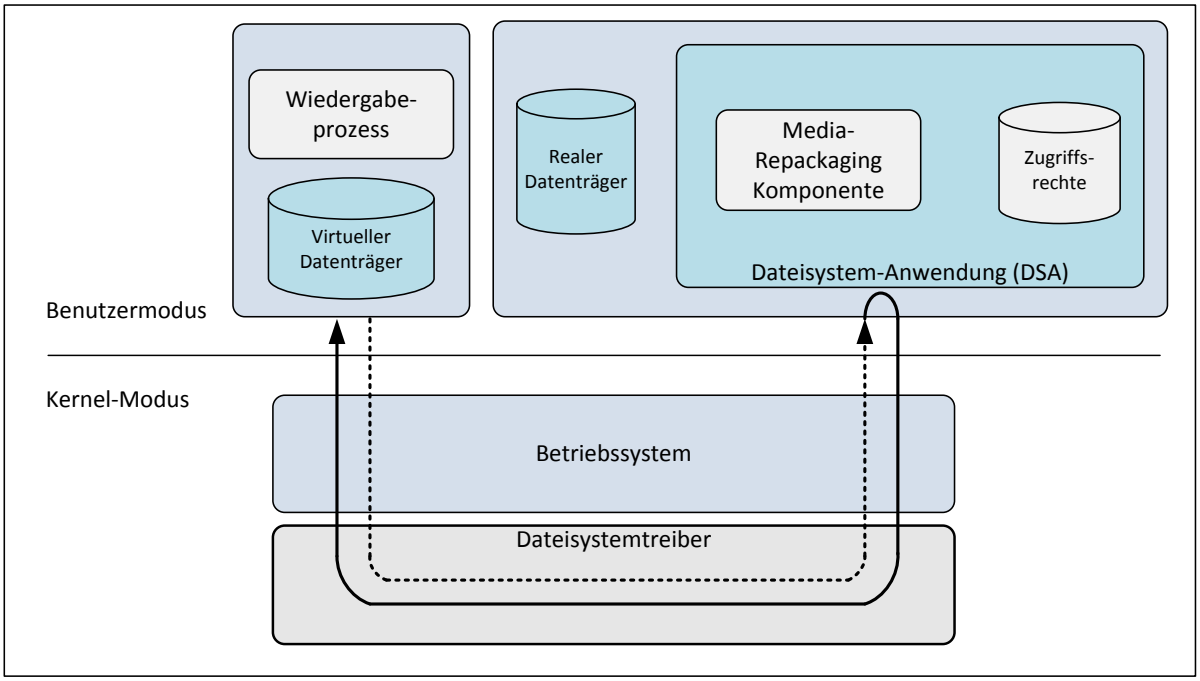

Abbildung 19: Architektur des Dateisystems für skalierbare Medien zur feingranularen Vergabe von Rechten auf skalierbare Dateien

Kernel-Modus und sorgen für die Übergabe der Aufrufe zwischen den vorgestellten Komponenten, die alle im sog. *User-Modus* des Betriebssystems ausgeführt werden.

Als Basis für diese Entwicklung wurde die Dokan Softwarebibliothek [33] verwendet, die eine Entwicklung virtueller Dateisysteme (vgl. Abschnitt 2.2.3) unter Windows erlaubt. Dabei fiel die Wahl auf Dokan, da es die ausgereifteste Implementierung eines virtuellen Dateisystems unter dem Windows-Betriebssystem darstellt, Windows als Entwicklungsplattform favorisiert wurde und die Implementierungen aus dem Unix-Umfeld keine signifikanten Vorteile für die Implementierung boten.

4.1.2 Arbeiten mit dem virtuellen Dateisystem

Die Ausführung und Nutzung des virtuellen Dateisystems für skalierbare Mediendaten lässt sich in folgende Punkte unterteilen:

1. Start und Initialisierung: Die DSA wird als ausführbare Datei gestartet. Als Übergabeparameter erhält das Programm den Laufwerksbuchstaben, der später im Dateibaum eingehängt wird. Nach dem erfolgreichen Start des DSA-Prozesses steht das virtuelle Dateisystem zur Verfügung. Als Einstiegspunkt bekommt der angemeldete Benutzer einen virtuellen Ordner mit seinem Benutzernamen angezeigt. Zugriff auf diesen Ordner erhält er durch eine ACL, die mit Hilfe der in der Datenbank gespeicherten Daten nach dem Start des Dateisystems erzeugt wurde. Innerhalb dieses virtuellen Ordners findet der Benutzer einen weiteren virtuellen Ordner für jeden realen Ordner, auf den er zugreifen kann.

2. Die Anfrage von Metadaten, wie z. B. die Anzeige aller Dateien in einem Ordner (engl. *directory-listing*) wird an den virtuellen Datenträger — und somit an das virtuelle Dateisystem — auf Seiten des Benutzers gestellt. Die Dokan-Bibliothek [33] sorgt für die Umleitung der Anfrage durch das Betriebssystem und den Dateisystemtreiber an die DSA. Hier werden, je nach Anfrage des Wiedergabeprozesses, die entsprechenden Dateioperationen, wie z. B. *OpenDirectory* oder *GetFileInformation* aufgerufen. Eine Rückkommunikation ist durch sog. *Callback*-Funktionen möglich. Mit Hilfe der Zugriffsrechte des angemeldeten Nutzers und den originalen Bilddaten können die gewünschten Informationen durch die MRK bereitgestellt und an den aufrufenden Prozess zurückgereicht werden.

3. Dateianfragen werden über die Anfrage *CreateFile* durch den aufrufenden Prozess initialisiert. Entsprechend dem Vorgehen bei der Anfrage von Metadaten wird die Anfrage an die DSA weitergeleitet. Anschließend werden auch hier die Benutzerrechte des aktuellen Nutzers auf die angefragte Datei geprüft. Muss auf Grund der Benutzerrechte eine Reduktion der Qualität und/oder Auflösung an der skalierbaren Datei vorgenommen werden, wird die Media-Repackaging-Komponente mit folgenden Parametern aufgerufen (vgl. Abbildung 20):

    a. Pfad zur originalen Datei,

b. prozentuale Angabe der Auflösungsreduktion des Videos/der Bildsequenz,

c. prozentuale Angabe der Qualitätsreduktion der skalierbaren Mediendatei.

Dabei werden die Quelldateien zur Reduktion einer Auflösungsstufe nicht neu encodiert, sondern lediglich eine reduzierte Anzahl von Datenpaketen der originalen Datei neu zusammengestellt und der Header entsprechend Abbildung 18 neu erstellt, so dass dieser zu der neuen Bildvariante passt.

4.1.3 Implementierung

Im Rahmen dieser Arbeit wurde ein Prototyp des vorgeschlagenen Dateisystems implementiert, der neben der gewünschten Funktionalität der Rechtevergabe auch als technische Grundlage für Untersuchungen dient, deren Details noch in weiteren Abschnitten dieser Arbeit betrachtet werden. Nachfolgend werden die wesentlichen Schritte bei der Implementierung mit Hilfe von Sequenzdiagrammen erläutert.

Das Zusammenspiel der Objekte während der Initialisierung des Dateisystems wird in Abbildung 21 gezeigt. Hierbei erhält der Benutzer die Möglichkeit, den gewünschten Laufwerksbuchstaben, sowie den Namen des Dateisystems in einer grafischen Benutzeroberfläche einzutragen. Zum Start des Mount-Prozesses betätigt der Anwender anschließend die entsprechende Schaltfläche. Das *MainWidget* gibt die gewählten Parameter an die Klasse *SFSCore* weiter. *SFSCore* bündelt hierbei die Basisfunktionalität der gesamten Applikation, wie z. B. die Verwaltung der Datenbankverbindung und die Administration der Log-Einträge.

Anschließend wird eine Instanz der Klasse *DokanCore* erzeugt. *DokanCore* ist u. a. für die Zuordnung der Funktionen verantwortlich, die später durch das selbst entwickelte Dateisystem übernommen werden. Listing 1 zeigt hierzu einen Ausschnitt aus der *runCore()*-Methode. Die Variable *p_DokanOperations* ist für die spätere Ausführung elementar und verfügt über einen Funktionszeiger für jede Dateioperation, die unter dem Windows-Betriebssystem durchgeführt werden kann und durch den Dokan-Dateisystemtreiber unterstützt wird. In Listing 1 werden einige der Funktionszeiger durch eigens implementierte Funktionen aus der Klasse *DokanFunctions* initialisiert. Dabei ist zu erkennen, dass das neue, virtuelle Dateisystem nicht die komplette Funktionalität unterstützt — entsprechend werden die Funktionszeiger mit *NULL* initialisiert, wie z. B. für die Funktion *CreateDirectory* oder *DeleteFile*.

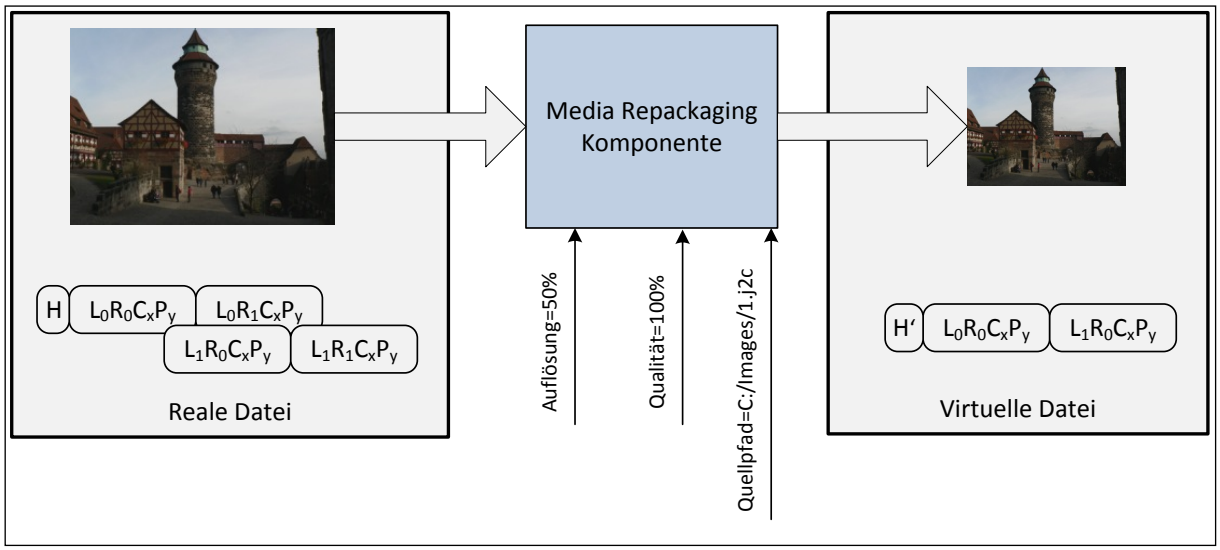

Abbildung 20: Auflösungsreduzierung einer JPEG 2000-Datei durch die Media-Repackaging-Komponente

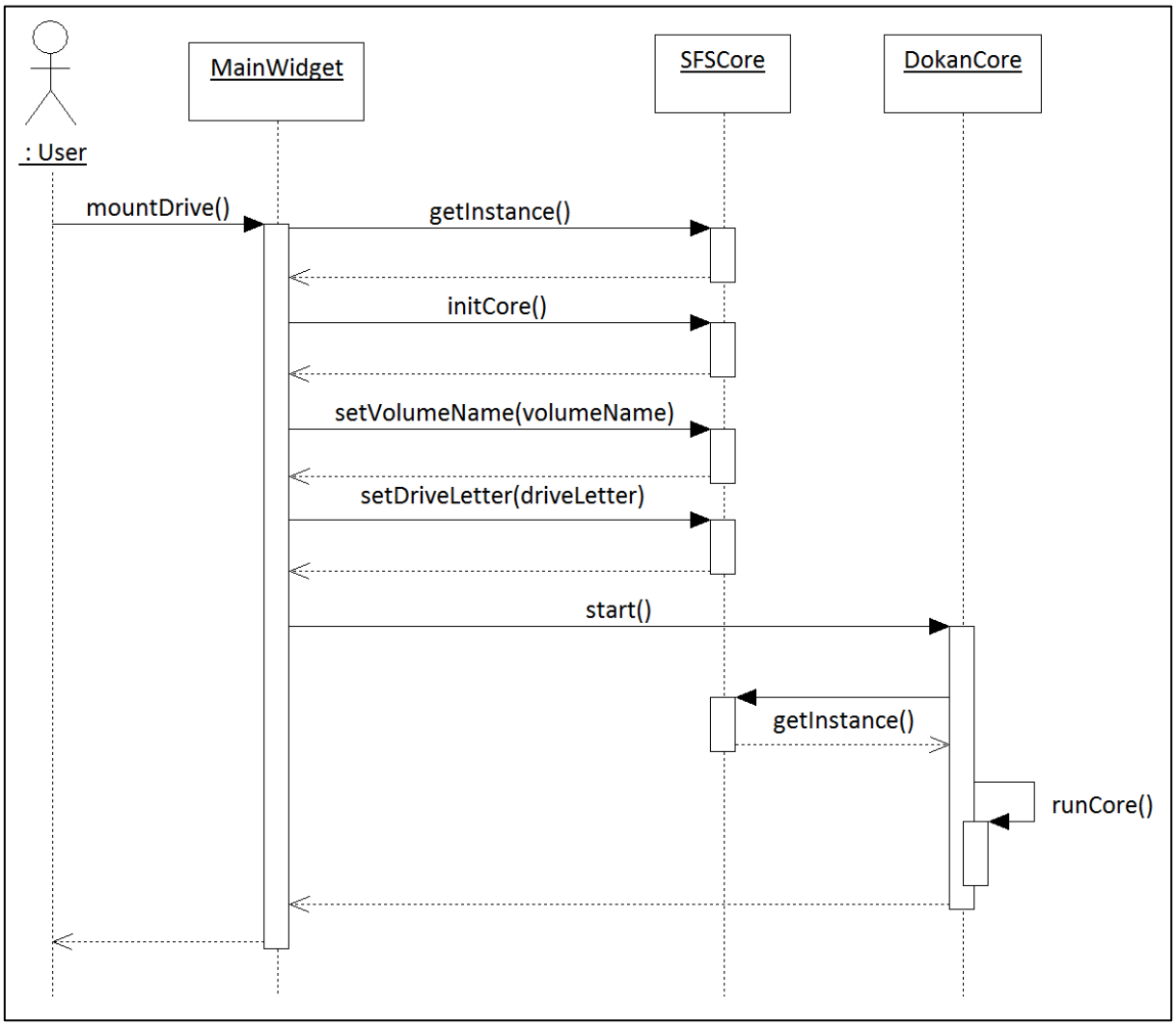

Abbildung 21: Sequenzdiagramm zum Mount-Prozess des virtuellen Dateisystems für skalierbare Medien

```
01:   (…)
02:   m_pDokanOperations->CreateFile = DokanFunctions::SFSCreateFile;
03:   m_pDokanOperations->OpenDirectory=
                              DokanFunctions::SFSOpenDirectory;
04:   m_pDokanOperations->CreateDirectory = NULL;
05:   m_pDokanOperations->CloseFile = DokanFunctions::SFSCloseFile;
06:   m_pDokanOperations->ReadFile = DokanFunctions::SFSReadFile;
07:   m_pDokanOperations->GetFileInformation=
                              DokanFunctions::SFSGetFileInformation;
08:   m_pDokanOperations->DeleteFile = NULL;
09:   m_pDokanOperations->DeleteDirectory = NULL;
10:   (…)
```

Listing 1: Ausschnitt aus der Initialisierungsroutine der Dokan-Optionen

Nach der Initialisierung der Dokan-Optionen wird die vorgegebene Bibliotheksfunktion *DokanMain* aufgerufen, die den Mount-Prozess abschließt und in einer Endlosschleife zur Abarbeitung ankommender Aufrufe endet. Hierdurch kann das Dateisystem verwendet werden. Üblicherweise listet ein Benutzer unmittelbar nach dem Mount-Prozess alle Dateien eines bestimmten Ordners auf. Das Zusammenspiel der Klassenobjekte für ein *directory-listing* zeigt das Sequenzdiagramm in Abbildung 22. Dabei sorgt üblicherweise der Dateimanager des Betriebssystems dafür, dass die Funktion *GetFileInformation* für jeden Ordner, sowie für jede Datei und Unterordner in diesem Ordner, aufgerufen wird. Entsprechend nimmt eine neu entwickelte, statische Funktion der Klasse *DokanFunctions* diesen Aufruf entgegen.

Da die Benutzer auf einem virtuellen Laufwerk arbeiten, erhält die Funktion *GetFileInformation* entsprechend virtuelle Pfadangaben, die im ersten Schritt in den realen Pfad übersetzt werden müssen.

Anschließend wird über die Klasse *UserManagement* ein geeignetes Plug-In für die Verarbeitung der ausgewählten Datei ermittelt. Der Vorgang ist in Abbildung 22 zur besseren Übersicht lediglich durch Aufruf der Funktion *getScalableMediaPlugin* und dem Rückgabewert eines Zeigers auf *JPEG2000ScalableMediaPlugin* gekennzeichnet. In Anhang A wird der detaillierte Ablauf dargestellt. Auch der Aufruf der Funktion *createScaledMediaFile* ist in Abbildung 22 zur besseren Übersicht nicht weiter ausgeführt, die fehlenden Objekte und entsprechende Funktionsaufrufe werden in Anhang B erläutert – im Erfolgsfall steht eine skalierte Datei zur Verfügung. Ein Aufruf an die Funktion *getFileSize* liefert die neue Dateigröße, die von der Funktion *GetFileInformation* benötigt wird. Dieser Wert kann jetzt an den Aufruf der Dokan-Bibliothek zurückgegeben werden. Dieser Vorgang wiederholt sich entsprechend für alle Mediendateien in einem Ordner.

Das Auslesen einer Datei beinhaltet zunächst ähnliche Ausführungsschritte, wie bspw. den Aufruf nach Dateiinformationen. Ein entsprechendes Sequenzdiagramm sowie die Funktionsweise der beteiligten Objekte ist in Anhang C darstellt. Innerhalb der Auslese-

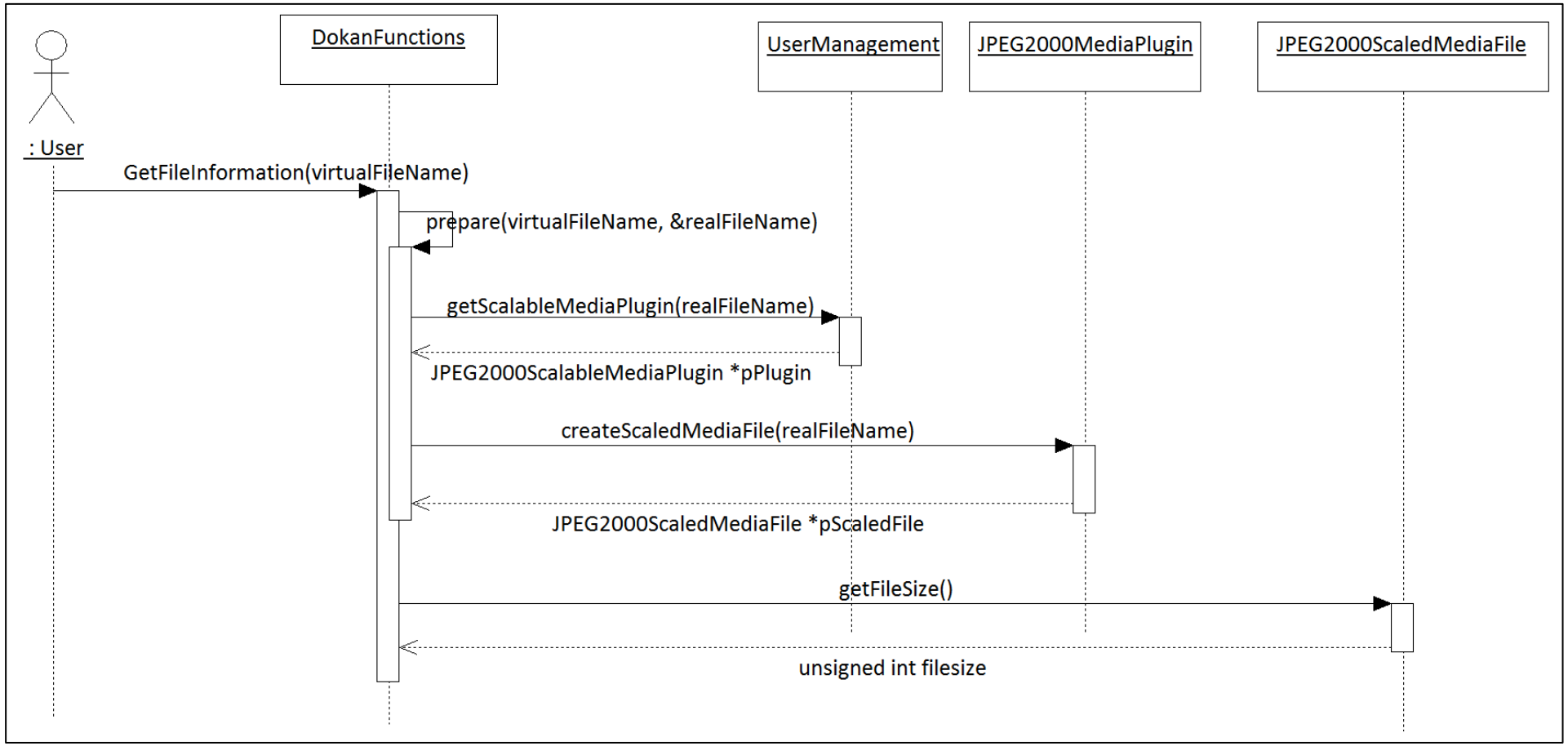

Abbildung 22: Sequenzdiagramm "Directory Listing" im virtuellen Dateisystem für skalierbare Medien

Routine kommt es dabei zu einer weiteren Caching-Stufe innerhalb des Dateisystems. Angefragte Dateivarianten werden dabei im Speicher des Benutzer-Prozesses vorgehalten, so lange genug Speicher zur Verfügung steht. Eine erste Implementierung des Dateisystems für skalierbare Medien unterstützt dabei die bekannte Cache-Verdrängungsstrategie First In First Out (FIFO). Weitere Teilarbeiten befassen sich mit Cache-Strategien, die speziell für skalierbare Medienformate entwickelt wurden. Diese werden im weiteren Verlauf dieser Arbeit genauer untersucht.

4.1.4 Funktionsweise

Nachfolgend soll die Funktionsweise des virtuellen Dateisystems für skalierbare Mediendaten erläutert werden. Zentrales Element des Systems ist eine grafische Benutzeroberfläche zum Einstellen der Zugriffsrechte auf die enthaltenen Dateien. Abbildung 23 zeigt dabei die aktuellen Rechte für den Benutzer *sbg*, der auf alle JPEG 2000-Bilder in voller Qualität, jedoch lediglich mit maximal 50% der verfügbaren Auflösung zugreifen kann. Ein zweiter Benutzer mit dem Namen *mrn* verfügt hingegen über die vollen Zugriffsrechte. Weitere Menüoptionen innerhalb des Prototypen erlauben u. a. die Vergabe eines Laufwerkbuchstabens, unter dem das virtuelle Dateisystem eingehängt wird, wie auch die Zuordnung zwischen realen Ordnern und deren virtuellen Namen innerhalb des virtuellen Dateisystems. Dabei wurde eine Zuordnung eines realen Ordners auf dem Laufwerk C vorgenommen, der eine JPEG 2000-Bildsequenz in 4K Auflösung enthält. Die durchschnittliche Dateigröße der realen Bilder der verwendeten JPEG 2000-Bildsequenz beträgt ~8MB.

Abbildung 24 zeigt die Auswirkung des entwickelten Dateisystems, wenn der Benutzer *sbg* auf den virtuellen Ordner mit dem Namen *JPEG2000_4K* zugreift. Es ist zu erkennen, dass dem Benutzer alle Bilder angezeigt werden. Dennoch ist die Dateigröße signifikant geringer im Vergleich zu den originalen Daten auf Laufwerk C. Dieser Umstand lässt sich mit der Reduktion der Auflösung erklären, da der aktuelle Benutzer die 4K-Bilder lediglich in der 2K-Version angezeigt bekommt. Darüber hinaus ist zu bemerken, dass der Benutzer *sbg* keine Kenntnis darüber besitzt, dass eine höherwertige Variante der Bilder existiert. Abbildung 25 gibt die Anzeige für den Benutzer *mrn* wieder, der die Bilder in voller Auflösung und Qualität erhält. Die angezeigten Dateigrößen sind hierbei identisch zu den Dateigrößen der realen Dateien.

4.1.5 Bewertung

Die feingranulare Rechtevergabe auf skalierbare Dateien erlaubt eine virtuelle Erstellung unterschiedlicher Varianten der vorhandenen Dateien. Analog zur Zugriffsvergabe auf Dateien lassen sich Daten somit vor unbefugtem Zugriff schützen. Ein mögliches Einsatzgebiet dieser Technologie ist bspw. die tägliche Verarbeitung (sog. *Dailies*) bei einer

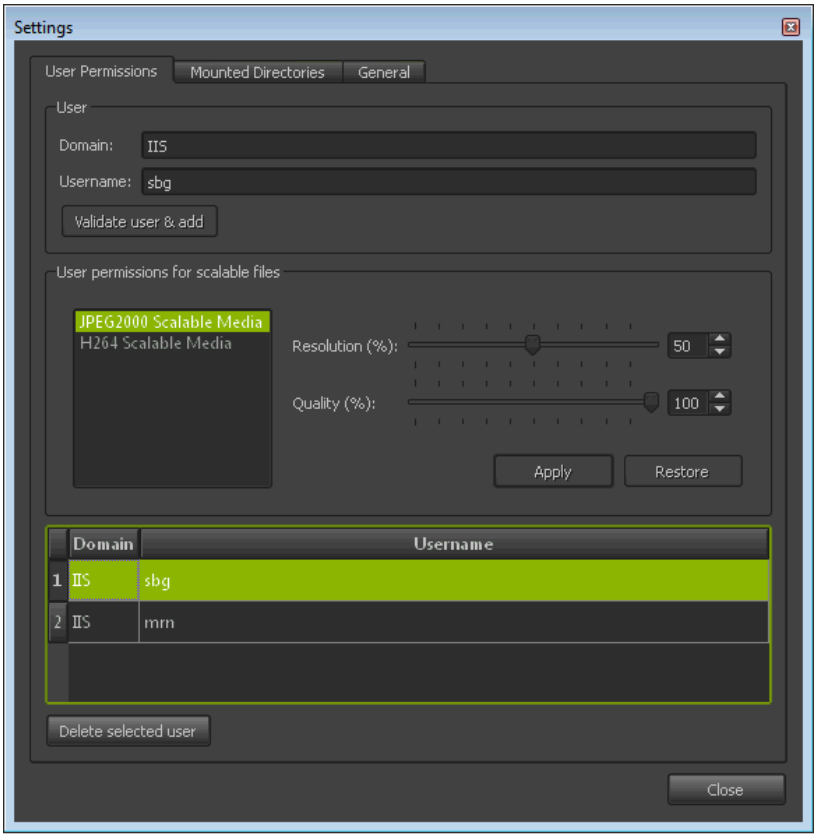

Abbildung 23: Benutzerschnittstelle zur Rechtevergabe auf skalierbare Medien am Beispiel von JPEG 2000, Detailbeschreibung im Fließtext

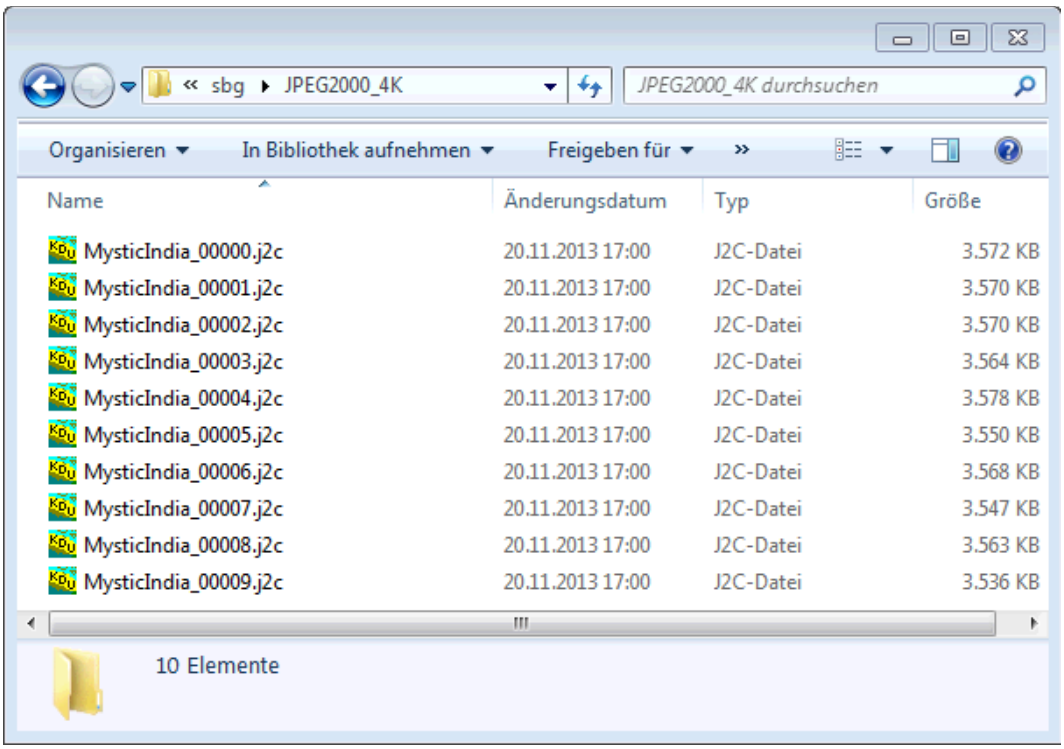

Abbildung 24: Anzeige einer JPEG 2000-Bildsequenz im Windows-Explorer für einen Benutzer mit eingeschränkten Zugriffsrechten. Die Dateigröße entspricht der 2K-Auflösungsversion der verwendeten JPEG 2000-Bildsequenz

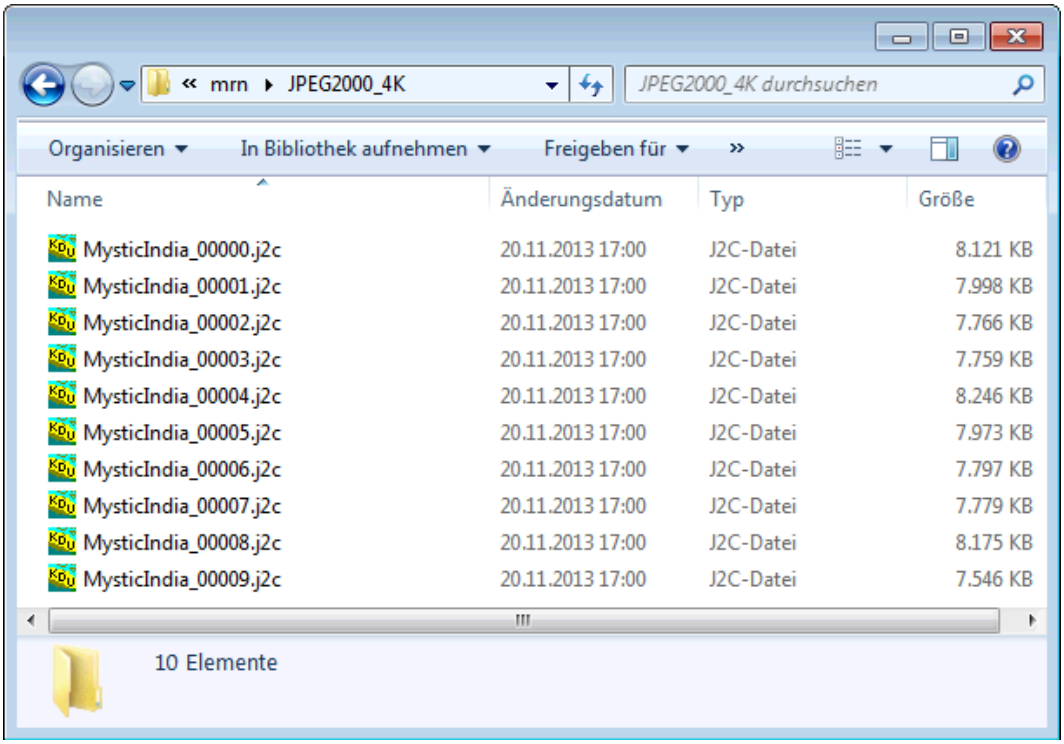

Abbildung 25: Anzeige einer JPEG 2000-Bildsequenz im Windows-Explorer für einen Benutzer mit uneingeschränkten Zugriffsrechten. Die Dateigröße entspricht der 4K-Auflösungsversion der verwendeten JPEG 2000-Bildsequenz

Filmproduktion. Hierbei werden jeden Tag mehrere Proxy-Varianten erstellt, die jene Filmszenen zeigen, die an diesem Tag gedreht wurden. Diese Varianten werden an verschiedene Personen versendet. Dazu gehört die Postproduktion, die üblicherweise eine Variante mit sehr guter Qualität und Auflösung bekommt, um dadurch bereits erste Muster für die Farbkorrektur erstellen zu können. Andere Personen hingegen bekommen bewusst nur eine Variante mit geringer Auflösung und/oder Qualität, um das Risiko der Piraterie so gering wie möglich zu halten. Durch das vorgestellte System ist es nicht mehr nötig, verschiedene Proxy-Varianten zu erstellen. Das ist im Besonderen interessant, wenn während der Produktion, z. B. als Codec in der Kamera, bereits ein skalierbares Format verwendet wird.

Ein weiteres, potenzielles Einsatzgebiet bieten Streaming-Plattformen für Audio- und Videodaten, die in einer großen Vielfalt im Internet vorhanden sind. Davon ausgehend, dass die Anbieter intern einen skalierbaren Media-Codec verwenden, lassen sich über das feingranulare Rechtemanagement verschiedene Geschäftsmodelle abbilden. Bspw. könnten verschiedene Abonnements angeboten werden, bei denen die maximal erhältliche Auflösung und/oder Qualität in Abhängigkeit des monatlichen Grundpreises variiert.

Durch die mögliche Vergabe von Rechten auf skalierbare Dateien kann die Erstellung von Proxy-Versionen reduziert werden, wenn verschiedene Benutzer unterschiedliche Varianten einer Datei angezeigt bekommen sollen. Neben den benötigten Ressourcen zum Erstellen dieser Proxies kann dabei vor allem Speicherplatz gespart werden. Um dies zu verdeutlichen, zeigt Abbildung 26 eine Messreihe, bei der ein potenzieller Speicherplatzbedarf zwischen

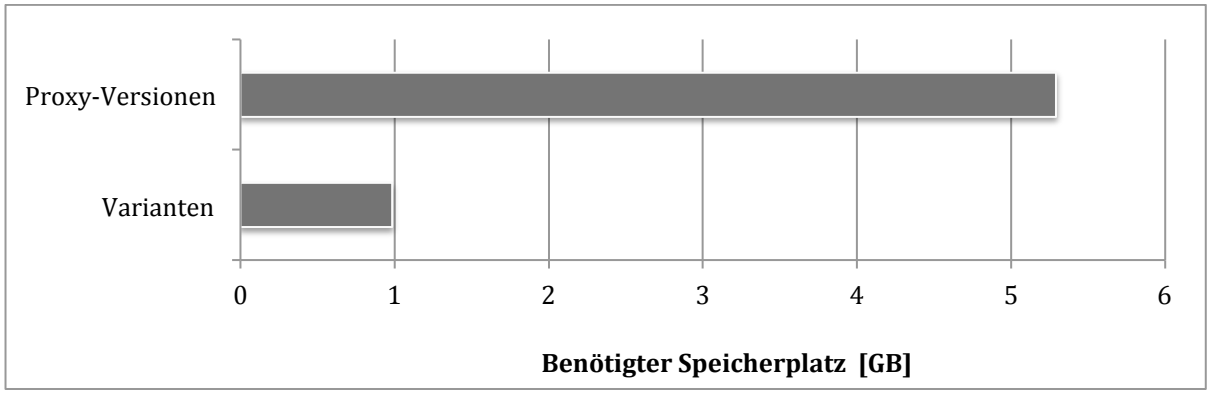

Abbildung 26: Vergleich des benötigten Speicherplatzes bei der Verwendung von Varianten skalierbarer Medien und Proxy-Varianten

Proxy-Varianten und unterschiedlichen Versionen skalierbarer Medien verglichen wurde. Hierzu wurde eine JPEG 2000-Testsequenz mit 120 Bildern, einer 4K-Auflösung, sechs Auflösungsstufen sowie drei Qualitätsschichten verwendet. Anhand der Bildparameter lassen sich bis zu 18 Varianten dieser Bilder erzeugen. Die 120 skalierbaren Bilder belegen zusammen etwa ~1GB Speicherplatz. Die entsprechenden 18 Proxy-Varianten belegen rund 5,4GB Speicherplatz. Auch wenn dieser Messwert sowohl von den verwendeten Bildern wie auch von der Anzahl der Varianten abhängt, wird deutlich, dass es zu signifikanten Ersparnissen bei der Speicherbelegung kommt, wenn weniger Proxy-Varianten erstellt werden müssen.

Weitere Messungen zeigen, dass die Gesamtleistung in Bezug auf die erreichte Durchsatzgeschwindigkeit des virtuellen Dateisystems etwa um einen Faktor 2-2,5 langsamer im Vergleich zu einem realen Dateisystem ist. Wie in Abschnitt 2.2.3 erläutert wurde, weisen virtuelle Dateisysteme zwei grundlegende Eigenschaften auf, die zu der reduzierten Durchsatzgeschwindigkeit im Vergleich zu einem realen Dateisystem führen. Das sind zum einen die Kontextwechsel zwischen Kernel- und Benutzer-Modus sowie häufige Prozesswechsel. Zum anderen werden die Daten mindestens ein weiteres Mal im Speicher kopiert. Auch wenn beide Operationen für eine einzelne Anfrage nicht teuer sind, summieren sich die Zeiten bei Anfrage von großen bzw. vielen Dateien auf.

Abbildung 27 zeigt die Messergebnisse von Leseanfragen auf eine JPEG 2000-Bildsequenz, die eine Dateigröße von rund 1 GB aufweist und über drei verschiedene Dateisysteme angesprochen wurde:

1. Lesen der kompletten Bildsequenz, die auf einer NTFS-formatierten Solid-State Festplatte[13] gespeichert ist (Abbildung 27: Reales Dateisystem) .

[13] Verwendet wurde eine SSD vom Typ Intel 335 Series 2,5-Inch 240GB SATA3.

2. Lesen der kompletten Bildsequenz von einem virtuellen Dateisystem mit einer simplen Spiegelungsfunktionalität (Abbildung 27: Spiegelung). Hierbei spiegelt das virtuelle Dateisystem einen Ordner von einem realen Dateisystem (hier: der Ordner von der SSD-Festplatte, der unter 1. angesprochen wurde) und bietet die enthaltenen Dateien unter einem neuen Laufwerksbuchstaben an. Anfragen an das virtuelle Dateisystem werden ohne Änderung an das reale System weitergegeben, respektive werden die Antworten ohne Manipulation an die aufrufende Applikation zurückgegeben.

3. Lesen der kompletten Bildsequenz vom entwickelten, virtuellen Dateisystem. Hierbei wurde dem lesenden Prozess (bzw. dem angemeldeten Benutzer, der den Prozess startet) voller Zugriff auf die vorhandene Bildsequenz gewährt, so dass die gelesenen Bilddaten identisch zu den Bildsequenzen unter 1. und 2. sind.

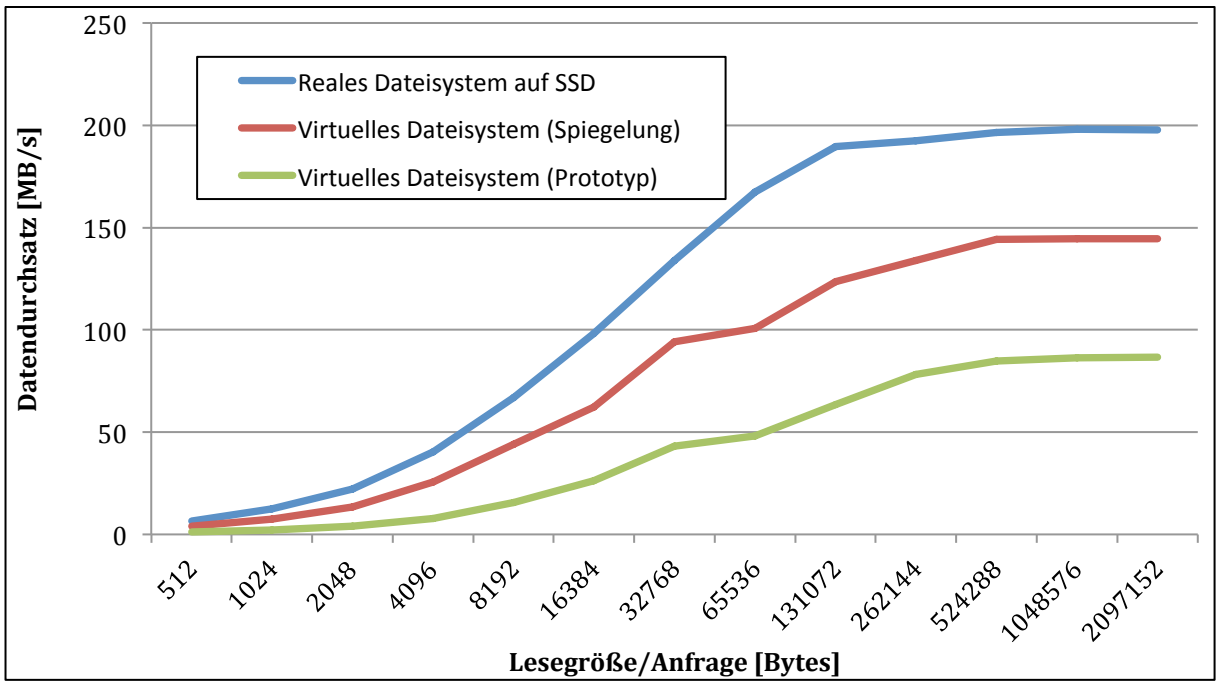

Abbildung 27: Gemessene Durchsatzraten beim Lesen einer Einzelbildsequenz vom realen und zwei virtuellen Dateisystemen

Bei der Messung wurde zusätzlich die angefragte Leseblockgröße pro Anfrage variiert. Je kleiner der Wert pro Leseanfrage, desto mehr Anfragen mussten pro Datei, respektive für die komplette Bildsequenz, gestellt werden. Es zeigt sich, dass dieser Wert ebenfalls einen Einfluss auf die Durchsatzleistung ausübt, allerdings einen ähnlichen Trend bei realen und virtuellen Dateisystemen zeigt. Zur Messung der Antwortzeiten der geschilderten Dateisysteme wurde eine vorhandene, interne Software verwendet, die speziell für diesen Einsatzzweck entwickelt wurde. Im Besonderen werden die Daten nach dem Empfang nicht weiter verarbeitet, sondern lediglich die ermittelten Transferzeiten ausgegeben.

Bereits zwischen dem realen und dem virtuellen Dateisystem mit der beschriebenen Spiegelungsfunktionalität zeigt sich ein Leistungsunterschied — das virtuelle Dateisystem ist

um etwa 35% langsamer. Da innerhalb des virtuellen Dateisystems keinerlei Verarbeitung der Daten vorgenommen wird, ist der Leistungsverlust auf die o. g. Gründe (Prozess-/Kontextwechsel und Datentransfer) zurückzuführen. Weiter zeigt sich, dass das virtuelle Dateisystem für skalierbare Mediendaten einen noch geringeren Datendurchsatz liefert. Dies ist durch die interne Bearbeitung der skalierbaren Mediendaten inkl. der Anwendung des Rechtemanagements zu erklären.

Die Leistung des entwickelten, virtuellen Dateisystems sollte sich zukünftig noch durch Optimierungen im Quellcode und eine weitere Reduktion der internen Kopieroperationen steigern lassen. Dieser Aspekt, sowie eine potenzielle Portierung in ein reales Dateisystem, sind dabei logische Schritte, die jedoch nicht innerhalb dieser Arbeit durchgeführt wurden.

4.2 Speicheradaptive Dateiorganisation

Datenspeicher gehören zu den wichtigsten Faktoren in vielen Phasen einer Film-Postproduktion, bei der verschiedene Komponenten eines Computersystems an ihre Leistungsgrenzen gebracht werden. Frühere Prozessoren waren bspw. nicht in der Lage, JPEG 2000-komprimierte Bildsequenzen in Echtzeit zu decodieren. Durch Nutzung schnellerer Prozessoren und Co-Prozessoren — wie bspw. Mehrkernprozessoren, dedizierte Beschleunigerkarten oder Grafikkarten — konnte dieser Engpass für heute gebräuchliche Auflösungen (HD und 2K) eliminiert werden. Einen weiteren Engpass stellen konventionelle Festplatten (HDDs) dar, die oft nicht in der Lage sind, die angeforderten Dateien unter Echtzeitbedingungen zu liefern. Zwar existieren schnellere Technologien, wie z. B. Solid State Drives (SSDs), doch konventionelle Festplatten bieten einen Kompromiss zwischen Schnelligkeit und wahlfreiem Zugriffsverhalten im Vergleich zu SSDs sowie bandbasierten Speichertechnologien und sind derzeit noch deutlich preisgünstiger als Solid-State-Technologien (vgl. 2.2.2).

Zur Eliminierung des Geschwindigkeitsengpasses, der durch Massenspeicher entsteht, werden in der Literatur u. a. zwei Lösungsansätze vorgeschlagen:

1. Verwendung eines RAID-Systems (vgl. 2.2.4) zur Erhöhung des Datendurchsatzes.
2. Erstellung sog. *Proxy-Dateien*, die üblicherweise eine geringere Auflösung und/oder Qualität als die Originaldateien aufweisen und anstatt dieser unter Echtzeitbedingungen abgerufen werden können.

Eine Nutzung der Skalierbarkeit moderner Kompressionsverfahren zur Echtzeitanlieferung durch verfügbare Speichersysteme wurde bisher kaum in Betracht gezogen. Ausgehend von den in Abschnitt 2.2.2 geschilderten Eigenschaften verschiedener Massenspeicher, werden an dieser Stelle optimierte Speicherverfahren für skalierbare Medien vorgestellt. Ziel ist es, die Skalierbarkeit derart mit den Eigenschaften der Datenträger zu kombinieren, dass ein erhöhter Durchsatz und/oder Echtzeitfähigkeit erreicht werden kann. Der erhöhte Durchsatz wird

durch eine intelligente Ablagestrategie der skalierbaren Daten beim Schreibvorgang auf HDDs erreicht. Dabei wird gezeigt, dass der Datendurchsatz von Festplatten mit rotierenden Scheiben signifikant erhöht werden kann, wenn die Mediendaten zuvor optimiert abgespeichert wurden. Echtzeitfähigkeit kann durch eine neue RAID-Konfiguration, die speziell für skalierbare Medien innerhalb dieser Arbeit entwickelt wurde, erreicht werden.

Ablagestrategien für skalierbare Medien auf Festplatten mit rotierenden Scheiben wurden bereits von Kang et al. in [53] sowie [128] untersucht. Hierbei liegt der Fokus auf Streaming-Anwendungen mit stark variierender Bandbreite zwischen Server und Klienten. Die Autoren stellen dabei das sog. *Harmonic-Placement* vor, bei dem häufig angefragte *Layer* eines skalierbaren Videos sequenziell auf der Festplatte abgelegt werden, damit der Aufwand zum Repositionieren der Leseköpfe bei Anfrage des Videos minimal bleibt. Weniger häufig angefragte *Layer* werden an einer separaten Stelle gespeichert. Weiter existieren relevante Vorarbeiten [13,14,15,69,70] auf dem Gebiet des vorrausschauenden Lesens (engl. *Prefetching*) der Daten, bei dem Informationen vom Klienten angefordert werden, bevor der Benutzer diese anfragt, um anschließend kürzere Reaktionszeiten des Gesamtsystems erzielen zu können. Diese Vorarbeiten werden im Folgenden beschrieben. In [15] wird die Betrachtung von skalierbaren Einzelbildern (z. B. Satellitenaufnahmen) untersucht, bei denen ein Benutzer in verschiedene Richtungen navigieren und zoomen kann. Es wird die Erkenntnis berücksichtigt, dass ein Benutzer, nachdem ein Bild angezeigt wird, eine gewisse Zeit benötigt, um die Daten zu analysieren und zu entscheiden, ob er weitere Daten für dieses Bild anfordern möchte. Ziel des vorgeschlagenen Algorithmus ist es, diese Entscheidungszeit zu nutzen, um weitere Bilddaten zu senden. Während der Entscheidungsphasen werden deshalb keine Bilddaten für den aktuell interessanten Bildausschnitt (engl. *Window of Interest* – WoI) nachgesendet, sondern Daten für mögliche, nachfolgende WoIs übermittelt. Diese Daten können z. B. nötig sein, wenn sich der Betrachter entscheidet, nach links/rechts zu navigieren. Die Simulationen in [15] zeigen, dass die Anwendung von Prefetching die Wartezeiten für nachfolgende Anfragen minimieren kann. Allerdings liegt die Schwierigkeit darin, die nächsten Anfragen des Nutzers vorherzusagen. Bereits sehr simple Navigationsmodelle, bei denen die Anzahl der möglichen Navigationsschritte stark limitiert ist, zeigen signifikant bessere Ergebnisse im Vergleich zu Verfahren, die ohne Prefetching arbeiten. In [13] bzw. [14] favorisieren die Autoren den Ansatz, so lange Daten für das aktuelle WoI zu senden, bis eine bestimmte Qualität erreicht wird. Das Navigationsmodell wird aus [15] übernommen und ein Reaktionsmodell eingeführt. Demnach setzt ein Nutzer das nächste Kommando nach t Sekunden ab, nachdem das Bild eine bestimmte Qualität q erreicht hat. Die Ergebnisse zeigen, dass das vorgestellte Prefetching-Verfahren nur dann von Vorteil ist, wenn zukünftige Anfragen in sehr hohem Maße vorhergesagt werden können und wenn der Nutzer eine sehr hohe Qualität des Bildes benötigt, um seine nächste Entscheidung zu treffen. Eine Kombination aus den Arbeiten [15] und [58] stellen die Autoren in [69] sowie [70] vor. Analog zu [58] soll ein JPIP-Proxy-Server eingesetzt werden. Da die Anfragen aktueller und zukünftiger WoIs das Netzwerk zu stark belasten würden, wird hier zusätzlich im Proxy ermittelt, welche

Anfrage an den Server die größte Qualitätssteigerung für aktuelle und zukünftige WoIs erreicht. Entsprechende Pakete werden anschließend angefragt und im Proxy-Server-Cache gespeichert. Angelehnt an die Verfahren von Lima et al. kann durch den Smart JPIP-Proxy-Server eine Steigerung bis zu 12 dB im Vergleich zum Standard-JPIP-Verfahren erreicht werden.

In [99] von Shunya et al. werden skalierbare Bild- und Video-Codecs im Bereich von Video-on-Demand-Systemen untersucht. Die Autoren plädieren für die Installation eines Caches, der logisch nah bei den Klienten installiert ist, damit eine Variante mit geringer Auflösung — die ggf. für eine Vorschau bereits angefragt wurde — wiedergenutzt werden kann, wenn das Bild bzw. Video in einer höheren Auflösung bzw. Qualität angefragt wird. Die Ergebnisse zeigen, dass große Cache-Speicher und skalierbare Codecs das Cache-Verhalten signifikant beschleunigen. Neben den Optimierungen auf Klient-Seite betrachten die Autoren in [57] serverseitige Aspekte, die — zusammen mit den Eigenschaften von TCP/IP — zur Verbesserung der Übertragung von JPEG 2000 dienen können. Senderseitig werden skalierbare Daten derart in einzelne Übertragungseinheiten zerteilt, dass die Unterteilung immer an Paketgrenzen eines JPEG 2000-Bildes vorgenommen wird, damit Daten — die üblicherweise zur unmittelbaren Qualitätssteigerung beim Empfänger dienen — komplett versendet werden. Hierzu wertet der vorgestellte sog. *Bandwidth Effective STreaming* (BEST)-Algorithmus die aktuelle Auslastung des TCP/IP-Netzes aus und teilt die Bilddaten entsprechend so auf, dass damit komplette Pakete der JPEG 2000-Datei gesendet werden und das Netzwerk nicht überlastet wird.

4.2.1 Anwendungsspezifische Anordnung von Informationspaketen

[14]Der Umgang mit skalierbaren Medien kann in zwei Arten unterteilt werden:

1. Der Anwender ignoriert die Möglichkeit der Skalierbarkeit und ruft die Daten immer in höchster Auflösung und Qualität ab.

2. Der Anwender nutzt die Skalierbarkeit und fragt Sub-Varianten an, um diverse Leistungsengpässe zu eliminieren. Diese Engpässe entstehen bspw. beim Datentransfer zwischen Speicher und Host-Computer oder durch zu langsame Decodierungseinheiten, die nicht in der Lage sind, die Datei in voller Auflösung und Qualität unter Echtzeitbedingungen wiederzugeben. Weiter ist es denkbar, dass die Arbeitsplätze nicht mit entsprechenden Monitoren ausgestattet sind, um die komplette Auflösung darzustellen. So werden Filme aktuell in 4K/Ultra HD aufgenommen und übertragen,

[14] Teilaspekte dieses Abschnitts wurden in [103] sowie [104] veröffentlicht.

obwohl die Monitore an den Arbeitsplätzen meist nur über eine Standard HD-Auflösung verfügen.

Nutzen Anwender die Eigenschaften der Skalierung, ist davon auszugehen, dass sie die gleiche Sub-Variante von Bildern oder Videos einer Sequenz und für die komplette Produktion anfragen werden. Weiter kann davon ausgegangen werden, dass andere Benutzer die gleichen Limitierungen an ihrem Arbeitsplatz vorfinden und dadurch ein ähnliches Zugriffsverhalten zeigen. Da die genannten Faktoren, wie z. B. Netzwerkdurchsatz, Bildschirmgröße oder Rechenleistung, innerhalb einer Produktionsstätte durch Kenntnis der verfügbaren Systeme im Voraus bekannt sind, kann ein Zugriffsverhalten der Nutzer auf skalierbare Medien gewissermaßen vorhergesagt werden. Glücklicherweise ist in diesem Szenario das Nutzerverhalten deutlich besser vorhersehbar als in den Arbeiten [69] sowie [70], da sich die Leistungsfähigkeit der verwendeten Hardwarekomponenten üblicherweise nicht ändert und Bilder in der Regel sequenziell abgespielt werden.

Durch das abschätzbare Nutzerverhalten lassen sich zwei wichtige Eigenschaften ableiten:

1. Wird nur eine Subvariante der skalierbaren Daten angefordert, muss auch nur diese Subvariante vom Speicher gelesen und übertragen werden. Dadurch ergibt sich bei konventioneller Speicherung, dass Bereiche der Datei beim Lesen übersprungen werden können, da diese nicht für die entsprechende Subvariante benötigt werden.

2. Tests zeigen, dass das Lesen von Festplatten maximale Transferraten liefert, wenn große Datenmengen sequenziell gelesen bzw. geschrieben werden können.

Zu 1:

Abbildung 28 zeigt drei verschiedene Szenarien zum Zugriff auf eine JPEG 2000-Sequenz. In Abbildung 28-1 wird die volle Auflösung und Qualität vom Dateisystem angefragt. Entsprechend werden alle zum Bild gehörenden Datenpakete zurückgeliefert. Hierbei ist zu beachten, dass die Daten derart organisiert sind, dass eine qualitätsoptimierte Speicherorganisation vorliegt. Nach Erhalt der Daten ergeben sich folgende Möglichkeiten: (i) Der aufrufende Prozess decodiert den kompletten Codestream. In diesem Fall muss das decodierte Bild anschließend skaliert werden, damit es vollständig auf dem HD-Display angezeigt werden kann. (ii) Der Decoder verwirft die übertragenen 4K-Anteile der Datei, decodiert lediglich die Subvariante und stellt diese am Computer-Display dar. In diesem Fall könnte der Datentransfer effizienter gestaltet werden, da die 4K-Datenpakete empfängerseitig nicht berücksichtigt werden.

In Abbildung 28-2 liest der Prozess die 2K-Subvariante ein. Auf Grund der Struktur der skalierbaren Einzelbilder kann die Festplatte die Daten nicht mehr ohne Unterbrechung lesen, was zu einer potenziellen Reduktion des Datendurchsatzes führen kann.

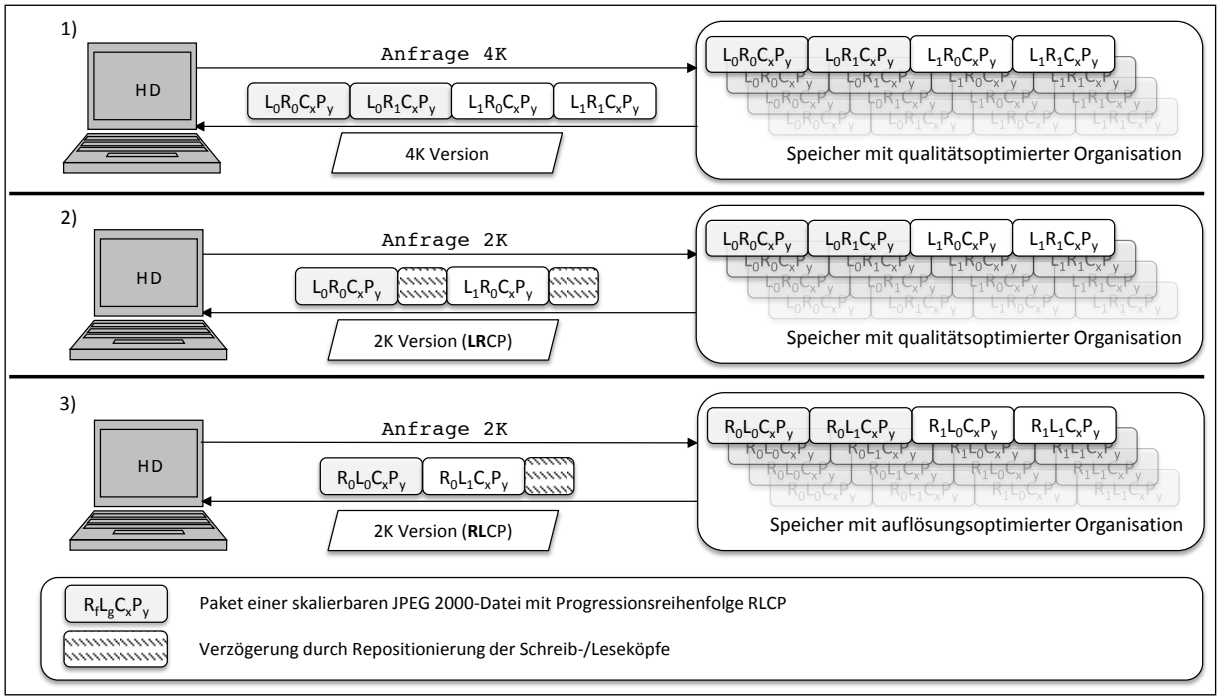

Abbildung 28: Grundprinzip bei der Anpassung der Progressionsreihenfolge für ein vorhersehbares Nutzerverhalten

In Abbildung 28-3 wurden die Daten von einer qualitätsorientierten zu einer auflösungsorientierten Progressionsreihenfolge umsortiert, so dass die 2K-Variante ohne Repositionierung der Schreib-/Leseköpfe gelesen werden kann. Es zeigt sich, dass die Datenpakete für die 2K-Auflösung wieder ohne Unterbrechungen gelesen werden können, was den Durchsatz einer Festplatte potenziell erhöhen kann. Diese Durchsatzsteigerung soll nachfolgend untersucht werden.

Zu 2:

Beispielhaft soll hier eine handelsübliche Festplatte[15] gezeigt werden, deren Lesegeschwindigkeit von einer entsprechenden Mess-Software [101] mit etwa 93 MB/s ermittelt wurde. Es galt zu zeigen, dass der erzielbare Datendurchsatz beim Auslesen einer Subvariante signifikant sinkt, wenn die Struktur der Quellbilder bestimmte Eigenschaften, wie z. B. eine für den Anwendungsfall nicht optimale Progressionsreihenfolge, aufweist.

Für die Messungen wurde eine generische JPEG 2000-Bildsequenz mit 120 Bildern, sechs Auflösungsstufen, fünf Qualitätsschichten und drei Farbkomponenten erstellt. Jedes Bild enthält somit neben dem Header 90 (6*5*3) Informationspakete, die Auflösung beträgt 4K (4096x2160) und die Progressionsreihenfolge ist qualitätsorientiert (LRCP). Für die Messungen wurden entsprechende Subvarianten mit den Auflösungen 0,5K (512x270)

[15] Seagate Barracuda Hard Drive ST3250310AS, 4 discs, 8 heads, 7200 RPM.

Bildpunkten, 1K sowie 2K von der Festplatte angefragt, die mit einem FAT32-Dateisystem formatiert wurde. Bei der 0,5K Variante werden pro Einzelbild 45 (3*5*3) Pakete gelesen und entsprechend die übrigen 45 Pakete übersprungen. Zur 1K Variante gehören 60 (4*5*3) Pakete, respektive 75 (5*5*3) Pakete zur 2K-Variante.

Tabelle 6 zeigt, dass die durchschnittliche Menge gelesener Daten pro Leseaufruf bei niedrigen Auflösungen gering ist. Entsprechend werden erwartungsgemäß größere Bereiche der Daten zwischen den Leseanforderungen übersprungen. Die reine Anzahl der Neupositionierungen der Festplattenköpfe ist jedoch bei allen untersuchten Subvarianten identisch, wodurch sich folgende Schlussfolgerung ergibt: Die Neupositionierung der Schreib-/Leseköpfe erzeugt einen gewissen Zusatzaufwand, der den Datendurchsatz der Festplatte verringert. Werden darüber hinaus während den Leseanfragen nur geringe Datenmengen abgerufen, ist die Festplatte im Wesentlichen mit der Neupositionierung beschäftigt und erreicht daher nur einen geringen Datendurchsatz. Einen Vergleich der Durchsatzmessungen zeigt Abbildung 29. Es wird deutlich, dass die Anfrage von Subvarianten dazu führt, dass der Datendurchsatz der Festplatte weit unter den eigentlichen Möglichkeiten bleibt. Beim Auslesen der 0,5K Variante konnte lediglich ein Durchsatz von rund 9 MB erreicht werden. Das entspricht etwa 10% der eigentlichen Durchsatzgeschwindigkeit der verwendeten Festplatte. Hier werden Subvarianten üblicherweise auf Grund von Hardwarebeschränkungen — die vorher bestimmbar sind — vom Datenträger angefragt.

Tabelle 6: Datenbereiche und Kennzahlen beim Einlesen verschiedener Subvarianten einer JPEG 2000-Sequenz mit Progressionsreihenfolge LRCP

Auflösung	**Gelesene Pakete**	**Übersprungene Pakete**	**Gelesene Bereiche [Paketindex]**	**Übersprungene Bereiche [Paketindex]**	**Neupositionierungen**	**Durchschnittliche Anzahl gelesener Daten/Bild**	**Durchschnittliche Anzahl übersprungener Daten/Bild**
0,5K	45	45	1-9 19-27 37-45 55-63 73-81	10-18 28-36 46-54 64-72 82-90	5	0,067 MB	1,61 MB
1K	60	30	1-12 19-30 37-48 55-66 73-84	13-18 31-36 49-54 67-72 85-90	5	0,22 MB	1,46 MB
2K	75	15	1-15 19-33 37-51 55-69 73-87	16-18 34-36 52-54 70-72 88-90	5	0,72 MB	0,96 MB

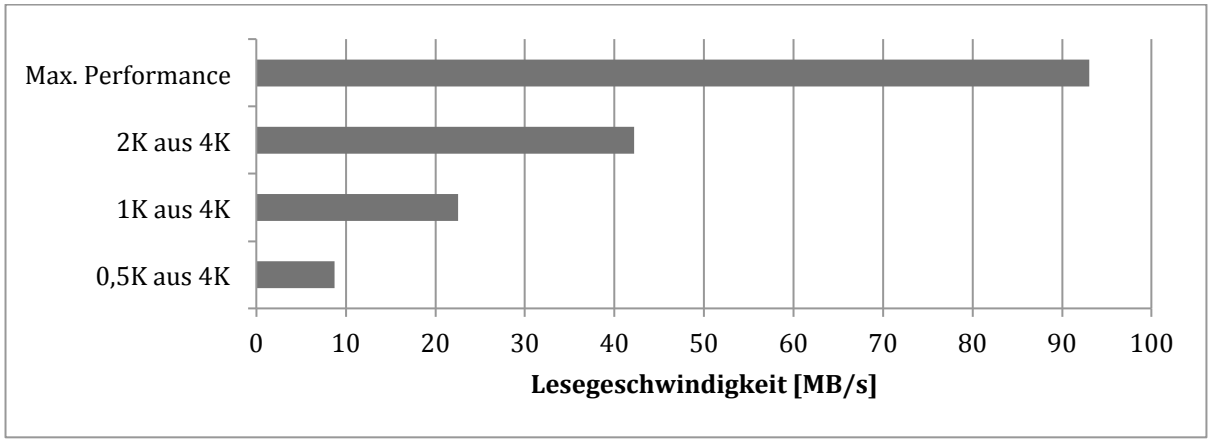

Abbildung 29: Erreichter Datendurchsatz beim Auslesen verschiedener Subvarianten (0.5K, 1K und 2K) einer JPEG 2000-Bildsequenz von einer Festplatte mit FAT32-Dateisystem im Vergleich zur maximal erreichbaren Lesegeschwindigkeit

Bei dem neuen Verfahren wird eine Änderung der Progressionsreihenfolge der skalierbaren Bilder beim Schreibvorgang auf die Festplatte vorgeschlagen, wenn diese nicht optimal für das erwartete Zugriffsverhalten ist. Abbildung 30 zeigt die dateiorientierte und sequenzorientierte Umstrukturierung von skalierbaren Einzelbildern. Abbildung 30-a zeigt dabei Optimierungen innerhalb einer Datei. Dieser Vorgang kann sowohl während des Beschreibens des Datenträgers, wie auch durch nachträgliche Umsortierung erfolgen, da die Dateigrößen — und der damit belegte Speicherplatz auf der Festplatte pro Datei — konstant bleiben. Zunächst liest ein entsprechender Algorithmus den Header (h_o) vom ersten Bild i_0. Hierdurch können die Progressionsreihenfolge und die Längen der einzelnen Pakete (α_0 bis δ_0) bestimmt werden. Sind die Datenpakete für das vermutete Nutzerverhalten, das sich im Wesentlichen aus den verfügbaren Hardwarekomponenten und Bildschirmgrößen der Arbeitsplätze ableitet, nicht optimal abgelegt, erfolgt eine Änderung der Progressionsreihenfolge. Hierzu werden die Pakete zunächst in anwendungsfallrelevante (α_0 und γ_0) und nichtanwendungsfallrelevante (β_0 und δ_0) Bereiche unterteilt. Bevor die Daten auf den Datenträger geschrieben werden können, müssen die Pakete umsortiert, der Header zu h_0' angepasst und der originale Header inkl. Paketindizes in einem Systemkatalog gespeichert werden, damit die Daten bei Bedarf wieder in die Originalstruktur überführt werden können. Der Vorgang wird entsprechend für alle Bilder der Eingangssequenz wiederholt. Wie in Abbildung 30-a zu erkennen ist, können anschließend alle anwendungsfallrelevanten Pakete ohne Unterbrechung gelesen werden. Zwischen den Bildern jedoch müssen die Schreib- und Leseköpfe neu positioniert werden.

Nachdem die Leseunterbrechungen für einen bestimmten Anwendungsfall innerhalb von Einzelbildern eliminiert wurden, sollen in einem weiteren Optimierungsschritt auch die Leseunterbrechungen zwischen den Einzelbildern verhindert werden. Neben der dateiorientierten Optimierung wird hierzu eine sequenzorientierte Optimierung durchgeführt (vgl. Abbildung 30-b). Dadurch kann eine komplette Sequenz ohne Unterbrechungen gelesen werden, was eine weitere Steigerung des Lesedurchsatzes mit sich bringt. Hierzu werden die

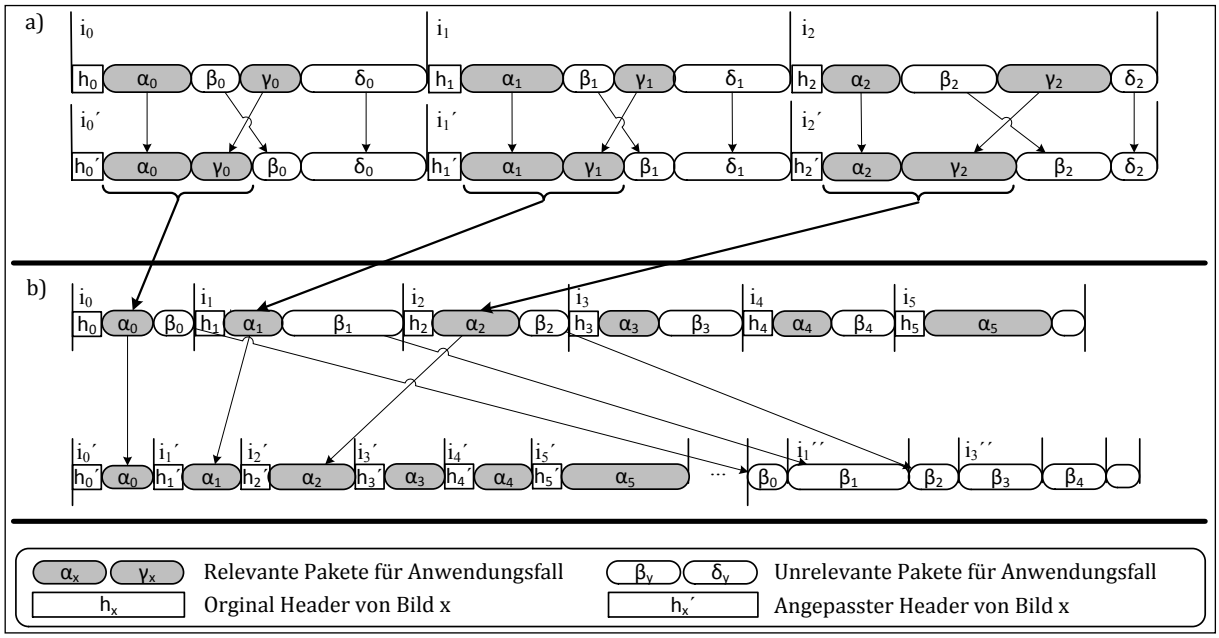

Abbildung 30: Dateiorientierte und sequenzorientierte Umstrukturierung skalierbarer Bilder a) Originale Bildsequenz mit drei skalierbaren Bildern i_0-i_2, die dateiorientiert so umsortiert werden, dass anwendungsfallrelevante Pakete nachfolgend ohne Unterbrechung gelesen werden können. b) Bereits dateiorientiert optimierte Bildsequenz mit sechs skalierbaren Bilder i_0-i_5, die sequenzorientiert so umsortiert werden, dass die anwendungsrelevanten Pakete einer ganzen Sequenz nachfolgend ohne Unterbrechung gelesen werden können

bereits dateiorientiert-optimierten Bilder in zwei Teildatenstränge geteilt. Erneut wird durch Interpretation des Headers ermittelt, welche Bereiche für einen Anwendungsfall relevant sind. Der Header h_0 wird für die neue Dateistruktur zu $h_0´$ angepasst. Anschließend wird die Datei aufgeteilt und die entsprechenden anwendungsfallspezifischen Pakete aller Bilder einer Sequenz sequenziell abgelegt. Der Speicherort der nicht-anwendungsfallspezifischen Pakete kann dabei variieren. Bspw. könnten diese Pakete im Anschluss an die Sequenz oder auf einem separaten Datenträger abgelegt werden.

Die in Abbildung 30-b gezeigte sequenzielle Ablagestrategie ist für die nicht-anwendungsfallspezifischen Datenpakete allerdings nicht zwingend erforderlich. Auch bei der sequenzorientierten Umsortierung der Datenpakete müssen die Eigenschaften der originalen Datei, sowie die neuen Speicherorte der Pakete, in einen Systemkatalog überführt werden, anhand dessen die originale Datei wiederhergestellt werden kann.

Testdaten und Implementation:

Um den Leistungsgewinn der beschriebenen Methoden zu ermitteln, wurden vier Testsequenzen einer qualitätsorientierten Einzelbildsequenz gemäß Abbildung 30 erstellt. Tabelle 7 gibt einen Überblick über die verwendeten Testsequenzen. Die Bilder der ersten Testsequenz beinhalten jeweils zwei Qualitätsschichten sowie Auflösungsstufen und drei Farbkomponenten. Damit verfügt jede Datei intern über 12 Pakete (Auflösungsstufen * Qualitätsschichten * Farbkomponenten).

Tabelle 7: Verwendete Testsequenzen zur Messung der Steigerung der Lesegeschwindigkeit unter Nutzung der anwendungsspezifischen Anordnung von Informationspaketen

#	Anzahl der Bilder	Auflösung	Progressions-reihenfolge	Durchschnittliche Dateigröße/Bild	Bemerkungen
1	120	4K	LRCP	9 MB	-
2	120	4K	RLCP	9 MB	Sequenz 1 nach Anwendung der dateiorientierten Optimierung
3	120	2K	RLCP	3,5 MB	Sequenz 2 ohne 4K Anteil gemäß sequenzorientierter Optimierung
4	120	1K	RLCP	1 MB	Sequenz 3 ohne 2K Anteil gemäß sequenzorientierter Optimierung
5	120	0,5K	RLCP	0,3 MB	Sequenz 4 ohne 1K Anteil gemäß sequenzorientierter Optimierung

Die Messungen wurden mit der bereits genannten Festplatte mit einer Kapazität von 250 GB durchgeführt. Um weiter ermitteln zu können, in welchem Ausmaß ein Dateisystem die Leistung des Gesamtsystems beeinflusst, wurden auf der verwendeten Festplatte vier leere Partitionen angelegt und jeweils mit einem der folgenden Dateisysteme formatiert: FAT32, ext2, ext3 und NTFS.

Den verwendeten Algorithmus zur Geschwindigkeitsmessung zeigt Listing 2. Hierbei wird zunächst der JPEG 2000-Header einer Quelldatei gelesen und die Dateistruktur interpretiert. Durch den Übergabeparameter *packetsToSkip* wird festgelegt, welche Pakete nicht gelesen werden sollen. Mit Hilfe der Struktur der Quelldatei und dem Wissen, welche Pakete übersprungen werden sollen, können anschließend noch die relevanten Pakete von der Festplatte gelesen werden. Die benötigte Zeit für das Auslesen der Sequenz wird dabei mittels der Systemzeit gemessen. Während der Messung wurden die Testsequenzen in der oben beschriebenen Reihenfolge sukzessive von allen Partitionen gelesen. Dieser Vorgang wurde 100-fach wiederholt. Anschließend wurden die jeweils 20 besten und schlechtesten Messergebnisse verworfen und der Mittelwert aus den verbleibenden Messungen gebildet. So wurde sichergestellt, dass die Messergebnisse nicht durch einsetzende Kernelprozesse, wie z. B. einem Indexdienst oder einem Viren-Scanner, verfälscht werden.

Der Zugriff auf den Datenträger wurde derart gewählt, dass das Betriebssystem keine Zwischenspeicherung der Daten veranlasst. Im Besonderen wurde das sog. *FILE_BUFFERING* im Windows Betriebssystem ausgeschaltet. Hierdurch kann sichergestellt werden, dass die Daten nicht beim zweiten Aufruf aus dem Zwischenspeicher geliefert, sondern erneut vom Massenspeicher gelesen werden.

```
01:  Algorithm ReadSequence (path, packetsToSkip)
02:    start = GetSystemTime();
03:    for all images img_i do
04:     j2kImg = ParseJpeg2000(img_i);
05:     relevantParts = GetRelevantParts(j2kImg, packetsToSkip);
06:     for all parts p_i do
07:       ReadPart(p_i);
08:     endfor
09:    endfor
10:    stop = GetSystemTime ();
11:    processingTime = (stop – start);
12:    return processingTime;
```

Listing 2: Pseudo-Code zur Zeitmessung verschiedener Dateiablagestrategien für skalierbare Einzelbilddateien

Die Messergebnisse zeigen, dass eine optimierte Ablage der Daten teilweise zu sehr hohen Leistungssteigerungen der verwendeten Dateisysteme führen kann. Hierzu zeigt Abbildung 31 beispielhaft die ermittelten Werte bei Abfrage der 0,5K-Subvariante aus einer JPEG 2000-Bildsequenz mit einer maximalen Auflösung von 4K. Dabei sind die verwendeten Bilder der Variante *0,5K aus 4K LRCP* ungünstig für eine Auflösungsskalierbarkeit angeordnet, dementsprechend werden nur kurze zusammenhängende Teile der Datei vom Massenspeicher gelesen, bevor die Schreib-/Leseköpfe der Festplatten neupositioniert werden. Hierbei zeigen die getesteten Dateisysteme einen Durchsatz von lediglich 9-10 MB/s. Dieser Wert lässt sich bereits durch die erste Optimierung — einer dateiorientierten Umsortierung der Datenpakete zu einer auflösungsorientierten RLCP-Progressionsreihenfolge — mit einem durchschnittlichen Durchsatz von etwa 26 MB/s um ca. 150% steigern. Eine weitere Verbesserung erreicht die sequenzorientierte Optimierung, wobei sich die Dateisysteme dabei unterschiedlich verhalten: Kann bei FAT32 eine weitere Steigerung auf rund 52 MB/s (das entspricht einer Durchsatzsteigerung von mehr als 400% im Vergleich zur nicht-optimierten Variante) erreicht werden, legen die ext-Dateisysteme — wie auch NTFS — nur gering, auf etwa 29 MB/s, zu. Ähnliche Ergebnisse liefert das Auslesen der 1K-Variante (vgl. Abbildung 32). Zunächst erfolgt auch hier eine signifikante Steigerung über alle Dateisysteme hinweg. Ohne Optimierung sind es etwa 22 MB/s, die durch die dateiorientierte Optimierung im Durchschnitt verdoppelt werden kann. Nach der sequenzorientierten Optimierung liefert FAT32 mit 62 MB/s erneut einen höheren Durchsatz als die anderen Dateisysteme (44-50MB/s). Vermutlich ist das FAT32-Dateisystem schneller, da es weder über ein Rechtemanagement noch über eine Journaling-Funktionalität (vgl. Abschnitt 2.2.3) verfügt, somit weniger Verwaltungsaufwand zu leisten hat und damit eine höhere Durchsatzrate erzielen kann. Beim Auslesen der 2K-Variante zeigt sich, dass die Steigerungen des Datendurchsatzes insgesamt geringer ausfallen. Das ist dadurch zu begründen, dass die Neupositionierungen immer weniger ins Gewicht fallen, wenn bei den einzelnen Leseanfragen große Datenmengen transferiert werden.

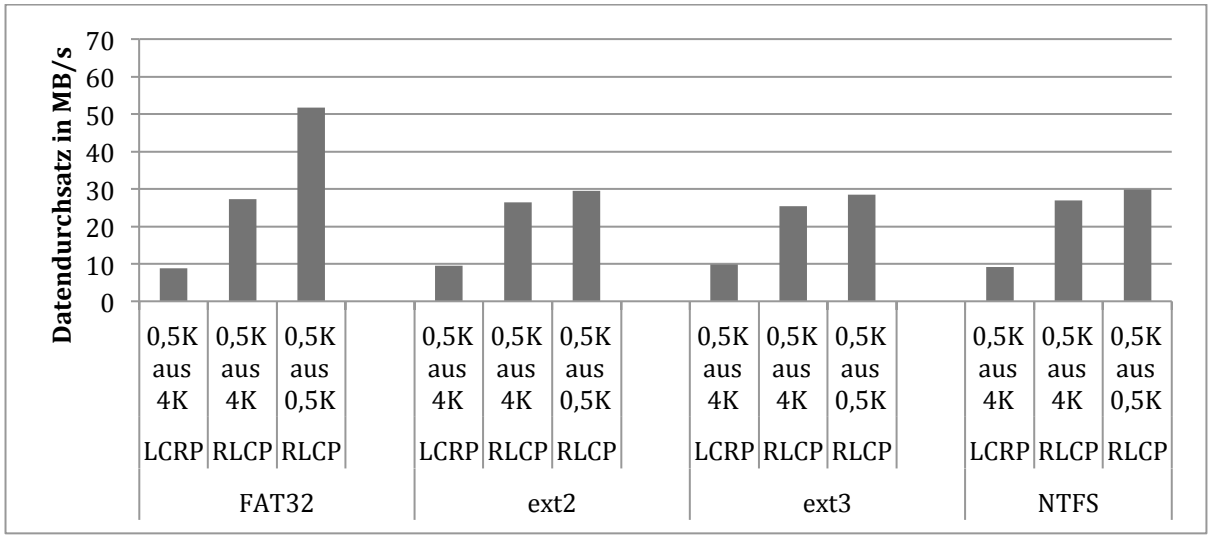

Abbildung 31: Datendurchsatz beim Auslesen der 0,5K-Variante aus einer 4K-Sequenz als Funktion der Progressionsreihenfolge von JPEG 2000-Bildern

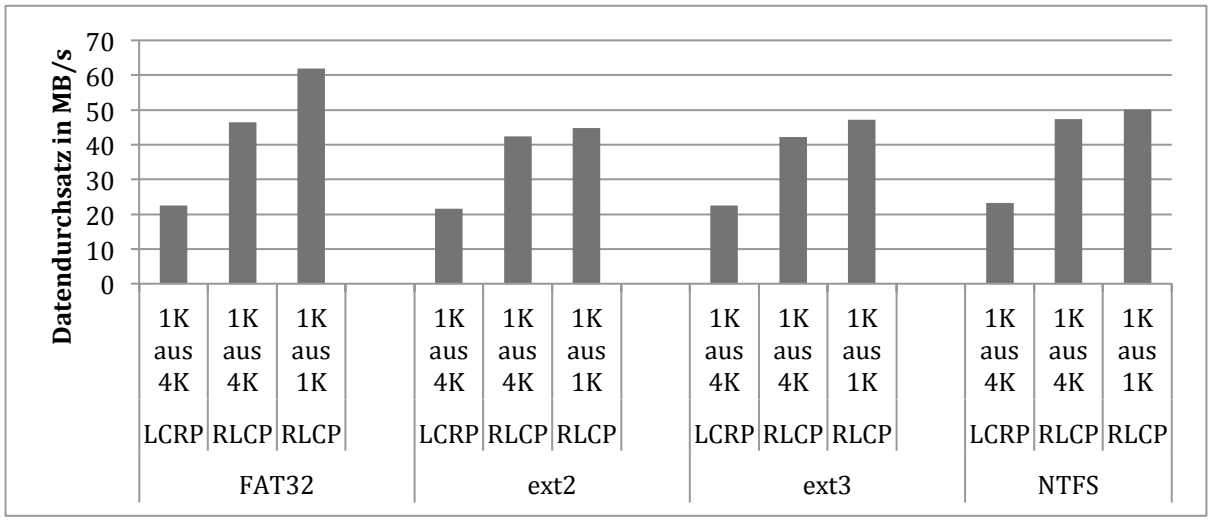

Abbildung 32: Datendurchsatz beim Auslesen der 1K-Variante aus einer 4K-Sequenz als Funktion der Progressionsreihenfolge von JPEG 2000-Bildern

Dennoch lässt sich auch hier eine Steigerung erzielen (vgl. Abbildung 33). Ohne Optimierung der Progressionsreihenfolge werden Durchsatzraten zwischen 44 MB/s (FAT32) und 48 MB/s (NTFS) erreicht. Die dateiorientierte Optimierung erreicht eine Steigerung auf 54 MB/s (FAT32) bis 61 MB/s (NTFS), dazwischen liegen ext2 und ext3 (59 MB/s). Mit 63 MB/s erreicht das NTFS-Dateisystem nach der sequenzorientierten Optimierung den Höchstwert der Messungen. Bei den ext-Dateisystemen ist hingegen keine Optimierung zwischen datei- und sequenzorientierter Optimierung mehr zu erzielen, der Durchsatzwert stagniert bei etwa 59 MB/s. FAT32 erreicht noch eine geringe Steigerung auf 56 MB/s, diese ist aber im Vergleich zu den anderen Subvarianten eher gering.

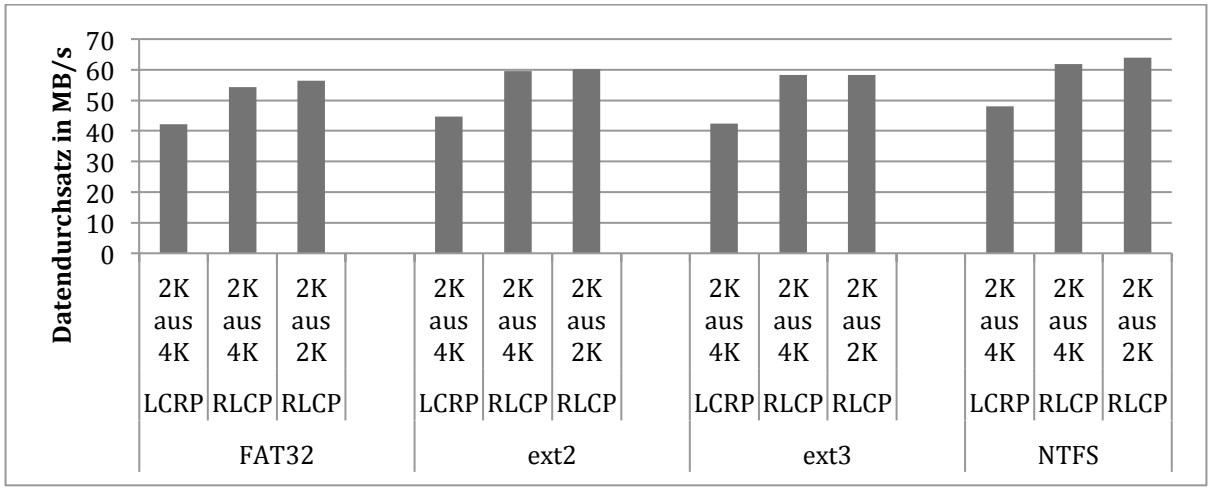

Abbildung 33: Datendurchsatz beim Auslesen der 2K-Variante aus einer 4K-Sequenz als Funktion der Progressionsreihenfolge von JPEG 2000-Bildern

Bewertung:

Die anwendungsspezifische Speicherung von Informationspaketen skalierbarer Medien liefert solide Ergebnisse, wenn das Zugriffsverhalten der Benutzer homogen ist und sehr gut vorhergesagt werden kann. Wie bereits geschildert, ist das ermittelte Zugriffsverhalten üblicherweise von den Hardware-Limitierungen entsprechender Arbeitsplätze abhängig. Diese Limitierungen können sich natürlich ändern, wodurch das Zugriffsmuster auf die skalierbaren Daten ebenfalls beeinflusst wird. Beispielhaft kann eine Änderung des Zugriffsmusters dadurch beeinflusst werden, dass alle Arbeitsplätze mit 4K-Displays ausgestattet werden, obwohl die Datenanbindung identisch bleibt. Die Benutzer werden in Folge dessen eine 4K-Variante der Bilder vom Massenspeicher anfragen und gegebenenfalls eine geringere Qualität der skalierbaren Datei nutzen, damit die Netzwerk- und Decodierungsressourcen weiterhin ausreichen, um die Bilder in Echtzeit abspielen zu können. Wurde also bisher eine auflösungsorientierte Progressionsreihenfolge bevorzugt, würde eine qualitätsoptimierte Auflösung jetzt bessere Ergebnisse erzielen.

Auf Grund der Struktur der skalierbaren Daten ist eine Neuordnung der Bilder mit relativ geringem Aufwand möglich, ohne diese neu codieren zu müssen. Bei der dateiorientierten Dateiorganisation können die Bilder sogar unter allen Umständen wieder an genau diejenige Stelle auf dem Datenträger geschrieben werden, an der die Bilder vor einer Neusortierung abgelegt wurden. Hierzu ist es lediglich notwendig, die vollständige Datei zu lesen und die Progressionsreihenfolge entsprechend des neuen Anwendungsfalls anzupassen. Hierbei werden die Pakete umsortiert und der Header entsprechend der neuen Progressionsreihenfolge angepasst. Anschließend kann die Datei wieder auf dem Datenträger gespeichert werden. Ggf. müssen die Metadaten in einem Dateisystemkatalog aktualisiert werden.

Allerdings erlauben es die verwendeten Dateisysteme nicht, bereits gespeicherte Dateien zu ersetzen. Auch wenn es für einen Anwender logisch so aussieht, als wenn eine Datei lediglich

ersetzt wurde, kann das Dateisystem die Daten auf einen völlig anderen Bereich der Festplatte kopiert haben. Noch ungünstiger ist der Fall, bei dem das Dateisystem nur Teile der Datei auf andere Sektoren verschiebt, da hierdurch bei einem anschließendem Leseaufruf mehrere Sprünge pro Bilddaten durchgeführt werden müssten.

Eine Neuordnung bereits gespeicherter Bilder wurde mit der RAW-Zugriffsmethode durchgeführt, da hierbei die Möglichkeit besteht, Datenpakete auf konkrete Sektoren eines Datenträgers zu schreiben. Dazu sollte ermitteltet werden, wie aufwendig die Neuorganisation der Datenpakete ist. Der hierfür verwendete Algorithmus *RestructureImg* ist in Listing 3 gezeigt. Hierbei werden drei verschiedene Zeiten gemessen: (i) die Lesezeit für die komplette Sequenz, (ii) die benötigte Zeit, um die Dateien im Speicher gemäß der neuen Progressionsreihenfolge (*newProgOrder*) umzusortieren sowie (iii) die benötigte Zeit für das Zurückschreiben der Einzelbilder.

Die Messungen für das Umstrukturieren einer bestehenden 4K-Bildsequenz analog zu Tabelle 8 von LRCP zu RLCP sowie CPRL wurden 100-fach durchgeführt und erneut die 20 besten und schlechtesten Ergebnisse entfernt. Es zeigt sich, dass die eigentliche Umsortierung der Daten mit durchschnittlich 4% nur einen geringen Anteil am Gesamtaufwand der Umstrukturierung ausmacht. Die Ergebnisse zeigen, dass eine nachträgliche Veränderung der Struktur im Wesentlichen als ein einzelner Lese- und Schreibvorgang pro Datei angesehen werden kann. Der Datendurchsatz der verwendeten Festplatte wurde während der Umstrukturierung mit ca. 92 MB/s gemessen.

```
01:  Algorithm RestructureImg (path, newProgOrder)
02:    readStart = GetSystemTime();
03:    j2kImg = ReadFile(rawDevice, startIndex, length);
04:    readStop = GetSystemTime();
05:    processingStart = GetSystemTime();
06:    j2kImg = ChangeProgOrder(j2kImg, newProgOrder);
07:    processingStop = GetSystemTime();
08:    writeStart = GetSystemTime();
09:    WriteFile(rawDevice, j2kImg, startIndex, length);
10:    writeStop = GetSystemTime();
11:    return (readStop - readStart) +
         (processingStop - processingStart) +
         (writeStop - writeStart);
```

Listing 3: Pseudo-Code zur Neusortierung der Progressionsreihenfolge nach Änderung des Zugriffsverhaltens inkl. Messung der benötigten Durchführungszeit

Tabelle 8: Vergleich der erforderlichen Bearbeitungszeiten für die wichtigsten Schritte bei der Reorganisation skalierbarer Medien

Prozess	**Prozentualer Anteil am Gesamtaufwand der Prozedur**
Lesen von der Festplatte	48%
Umsortierung der skalierbaren Dateien	4%
Zurückschreiben auf die Festplatte	48%

Weitere Optimierungsmöglichkeiten:

Die vorgestellten Verfahren zeigen, dass die Ausnutzung der Skalierbarkeit den Datendurchsatz von Festplatten deutlich steigern kann. Die vorgeschlagenen Algorithmen bieten eine solide Grundlage für die Integration in ein Dateisystem für skalierbare Medien (vgl. auch Abschnitt 4.1). Es sind noch weitere Optimierungen denkbar. Nachfolgend sind hierzu einige Punkte ohne Anspruch auf Vollständigkeit aufgelistet:

- Es wurde gezeigt, dass skalierbare Einzelbilder im JPEG 2000-Format zunächst auf einer Festplatte gespeichert und anschließend so umsortiert werden, dass die Daten für ein bestimmtes Zugriffsverhalten „optimal“ sind. D. h., die Bilddaten lassen sich bzgl. einer gewünschten Qualitäts-/ und Auflösungsvariante ohne große Zeitverzögerung, die durch die Bauform einer Festplatte hervorgerufen wird, auslesen. Diese Einzelschritte lassen sich auch kombiniert ausführen. In diesem Fall wird vor der initialen Speicherung der Bilddaten auf der Festplatte die gewünschte Progressionsreihenfolge erstellt und die Daten optimiert abgespeichert.

- Eine Neusortierung oder Rückgewinnung von Bilddaten, die bereits sequenzoptimiert abgelegt wurden, bedarf eines höheren Aufwands im Vergleich zur dateioptimierten Ablage. Besonders, wenn alle Teile der Datei auf einer Festplatte gespeichert wurden, können nachträgliche Kopierprozesse sehr aufwendig werden. Per Design liegen dabei relevante und nicht relevante Bereiche weit voneinander entfernt. Wird eine Bildsequenz in die Originalvariante zurückgeschrieben, müssen die Leseköpfe den relevanten Teil im Bereich A_r lesen, bevor der nicht-relevante Teil der Datei im Bereich A_n gelesen wird. Für das darauffolgende Bild wird wieder der Bereich A_r genutzt, bevor die nicht-relevanten Pakete des zweiten Bildes wieder im Bereich A_n gelesen werden usw. Ein entsprechendes Verfahren mit Kenntnis dieser Problematik könnte den Kopierprozess verbessern, indem zunächst alle relevanten Teile der Sequenz gelesen werden und am Ende jeder Datei ausreichend Platz gelassen wird, um die nicht-relevanten Teile der Datei anzuhängen. Anschließend fügt der Kopierprozess die nicht-relevanten Daten an den freien Stellen ein. Hierdurch muss auf der Quellplatte nicht der Lesebereich von A_r auf A_n gewechselt werden.

- Eine Umsortierung der Datenpakete beschleunigt ein bestimmtes Zugriffsverhalten auf die Bilddaten. Hier können Anwender abwägen, welche Progressionsreihenfolge die Dateien erhalten sollen. Kommt auf dem Speichermedium jedoch eine RAID-Variante zur Ausfallsicherheit hinzu (vgl. Abschnitt 2.2.4), können durch Entwicklung neuer RAID-Varianten verschiedene Progressionsreihenfolgen abgelegt werden. Ein Beispiel ist die Verwendung einer RAID-1-Konfiguration. Anstatt die Daten nur zu spiegeln, können diese mit unterschiedlichen Progressionsreihenfolgen auf jeweils einer der beiden Festplatten abgelegt werden. Der RAID-Controller entscheidet bei einer Anfrage, von welcher der beiden Festplatten die Daten anschließend schneller gelesen

werden können. Verfahren, wie diese Information an den RAID-Controller weitergeleitet werden können, werden in Abschnitt 5.2 vorgestellt.

4.2.2 Adaptive RAID-Konfiguration für skalierbare Medien

[16]Stellen die verwendeten Speichersysteme einen Engpass in der Datenverarbeitungskette dar, existieren verschiedene Verfahren, diese Engpässe zu beseitigen. Wichtig dabei ist die Erkenntnis, dass Echtzeitausspielung eine sehr wichtige Anforderung beim täglichen Umgang mit Bildsequenzen insb. im Bereich der professionellen und semiprofessionellen Filmproduktion ist. Gängige Verfahren zur Umgehung des Engpasses sind dabei u. a.:

1. Erstellung von Proxy-Dateien vor der Echtzeitbearbeitung (vgl. Abschnitt 2.3).
2. Umkopiervorgänge auf ein schnelleres Speichersystem vor der Bearbeitung des Filmmaterials. Dies ist z. B. der Fall, wenn Filmdaten zur Langzeitarchivierung auf Bandspeicherlaufwerke (Tape-Libraries) gesichert werden. Diese besitzen die Eigenschaft, dass der Datendurchsatz — im Vergleich zu einer Festplatte — gering ist und kein wahlfreier Zugriff auf die Daten möglich ist.
3. Zusammenschluss mehrerer Festplatten zu einem RAID-System, z. B. in einer RAID-0-Konfiguration (vgl. Abschnitt 2.2.4). Durch die Bündelung mehrerer Datenträger wird eine virtuelle, schnellere Festplatte erzeugt, die üblicherweise eine signifikant höhere I/O-Leistung aufweist. Hierbei entfällt die Erstellung von Proxydaten, da die Originaldaten entsprechend direkt bearbeitet werden.

Eine Betrachtung der bekannten RAID-Verfahren zeigt, dass diese über keinerlei Kenntnis von Metainformationen und Inhalt der Daten verfügen, die sie verarbeiten. Für nicht skalierbare Daten ist das auch nicht nötig, da eine Sub-Variante einer nicht-skalierbaren Datei keine sinnvollen Daten für eine nachgeschaltete Decodierungseinheit liefert. Wird bei der Aufteilung einer skalierbaren Datei allerdings deren interne Struktur berücksichtigt und Eingangsdaten entsprechend aufgeteilt, könnten genug Daten auf dem ersten Datenträger eines RAID-Systems gespeichert sein, um eine Vorschau der Bilddaten zu ermöglichen. Bei der Aufteilung an den Paketgrenzen skalierbarer Dateien muss allerdings berücksichtigt werden, dass die Daten für eine höhere Qualität/Auflösung von den unteren Varianten abhängig sind. In Abbildung 30-a muss bspw. das Datenpaket α_0 vor β_0 decodiert werden, β_0 vor γ_0 usw. Demnach muss zwangsläufig gewährleistet sein, dass jeder Datenträger, der Basisinformationen für weitere Pakete speichert, diese auch unter Echtzeitbedingungen liefern kann. Diese Eigenschaften können durch die verfügbaren RAID-Verfahren nicht gewährleistet werden, da durch klassisches *Striping* bereits Informationen der Basispakete über zwei

[16] Teilaspekte dieses Abschnitts wurden in [104] sowie [106] veröffentlicht.

oder mehrere Datenträger verteilt werden. Deshalb wurden zwei neue RAID-Konfigurationen speziell für die Speicherung skalierbarer Medien auf Datenträgerverbünden entwickelt und untersucht.

Abbildung 34 zeigt die bekannte RAID-0-Variante sowie die neu entwickelten, adaptiven RAID-Varianten, genannt *ARAID*. Abbildung 34-1 zeigt dabei, dass die Eingangsdaten in sog. *Stripes* unterteilt werden. Es ist deutlich zu erkennen, dass die Struktur nach dem Speichern der Stripes auf die angeschlossenen Datenträger nicht mehr vorhanden ist. Das führt dazu, dass der erste Datenträger (*Disk a*) nicht ohne *Disk b* verwendet werden kann. Abbildung 34-2 zeigt die erste ARAID-Variante, die für eine Echtzeitausspielung gedacht ist. Dabei werden die erreichbaren Lesegeschwindigkeiten aller angeschlossenen Datenträger vorab gemessen und zur Parametrierung des ARAID-Controllers verwendet. Die dabei verwendete Bildwiederholrate ist ebenfalls parametrierbar. Im Vergleich zur RAID-0-Variante ist die Struktur nach der Aufteilung erhalten geblieben. Besonders *Disk a* ist ohne *Disk b* ein vollwertiger Datenträger, mit dem es möglich ist, eine Vorschau auf die Bilddaten zu bekommen. Durch Anschluss von *Disk b* werden die bereits gelesenen Bilddaten je nach Struktur der skalierbaren Bilddatei in Auflösung und/oder Qualität verbessert. Quelldaten, die später unter Echtzeitbedingung nicht gelesen werden können, werden bei der ARAID-1-Variante nicht berücksichtigt. Genau hier ist auch der Unterschied zur Variante ARAID-2, die in Abbildung 34-3 gezeigt wird. Auch hier wird sichergestellt, dass die Datenträger die Daten später unter Echtzeitbedingung lesen können, allerdings wird der letzte Datenträger mit den verbleibenden Datenpaketen bespielt. Hierdurch ist es möglich, die Originaldaten komplett zurückzugewinnen.

Beim Auslesen der gespeicherten Daten kommt es zwangsläufig zu der Situation, dass nicht die komplette Datei vom potenziell langsamen Massenspeicher zurückgelesen werden kann. Allerdings steht dem ARAID-Controller die Header-Information zur Verfügung, anhand derer die Struktur der skalierbaren Datei nachgebildet werden kann. Mit Hilfe der Substitutionsmethode (vgl. Kapitel 3) ist es anschließend möglich, fehlende Dateneinheiten zu substituieren. Dieser Vorgang sollte im Controller stattfinden, da der Vorgang für den anfragenden Prozess somit vollkommen abstrakt ist.

4.2.3 Anwendung der Verfahren mit Bandlaufwerken

Die beschriebenen Verfahren zur Nutzung eines speziellen RAID-Systems für skalierbare Dateiformate können darüber hinaus auch ohne einen Verbund von Massenspeichern genutzt werden. Hierbei werden verschiedene Teile einer skalierbaren Datei auf zwei oder mehrere Bereiche des Datenträgers geschrieben. Dieses Verfahren kann vor allem bei der Verwendung von Bandlaufwerken mit geringer Lesegeschwindigkeit dazu genutzt werden, um eine Echtzeitausspielung der Daten mit reduzierter Auflösung/Qualität zu erreichen. Bisher konnte

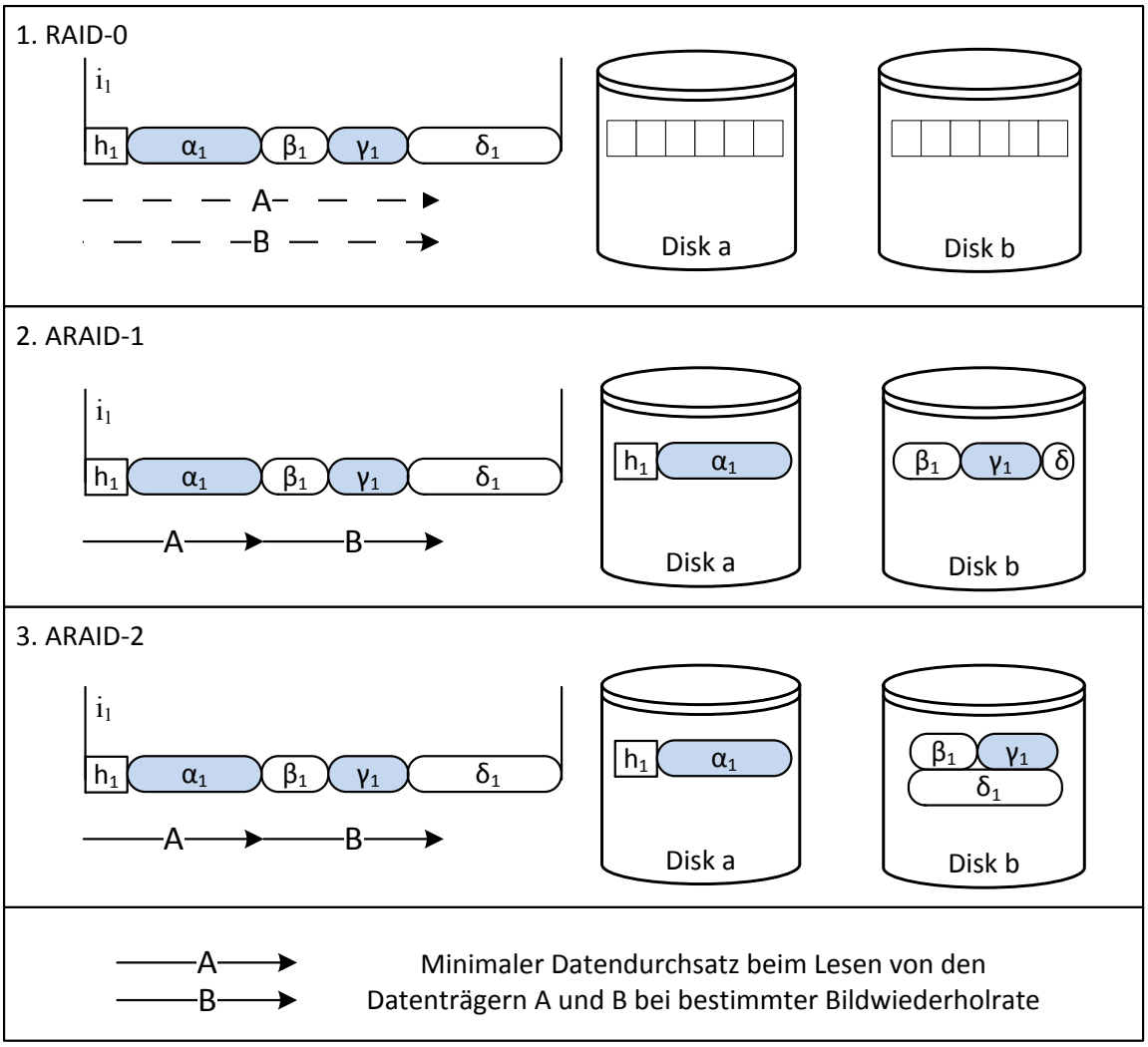

Abbildung 34: Vergleich verschiedener RAID-Verfahren: 1. RAID-0 mit Striping Eigenschaft des RAID-Controllers, 2. Adaptive RAID-1 (ARAID-1) Variante mit gewünschtem Datenverlust beim Schreiben, 3. ARAID-2 Variante, bei der die übrigen Daten auf den letzten Datenträger geschrieben werden

die Eigenschaft der Skalierung nur bedingt für Speicher mit sequenziellem Zugriffsverhalten genutzt werden, da es für eine effektive Nutzung der Skalierbarkeit nötig ist, zumindest in gewissem Rahmen — d. h. mit möglichst geringem Zeitverlust — über Datenblöcke hinwegspringen zu können. Gerade bei Bandlaufwerken sind diese Operationen, im Vergleich zum Lesen und Schreiben, sehr zeitintensiv und verhindern dadurch, dass skalierbare Medien hier sinnvoll eingesetzt werden können.

Abbildung 35 zeigt den Ablauf des Beschreibens und Auslesens von skalierbaren Medien auf einem Bandspeicher unter Nutzung der ARAID-2 Methode: Eine Sequenz skalierbarer Einzelbilder wird durch einen Controller eingelesen. Zuvor wurde auch hier die Schreib- bzw. Lesegeschwindigkeit des verwendeten Speichers ermittelt, so dass der ARAID-Controller die Eingangsdaten in zwei Kategorien aufteilen kann: (i) echtzeitfähiger Anteil der Datei bei vorgegebener Bildwiederholrate und (ii) nicht echtzeitfähiger Anteil. Der echtzeitfähige Teil der Daten wird direkt auf das Speicherband geschrieben, der nicht-echtzeitfähige Anteil auf

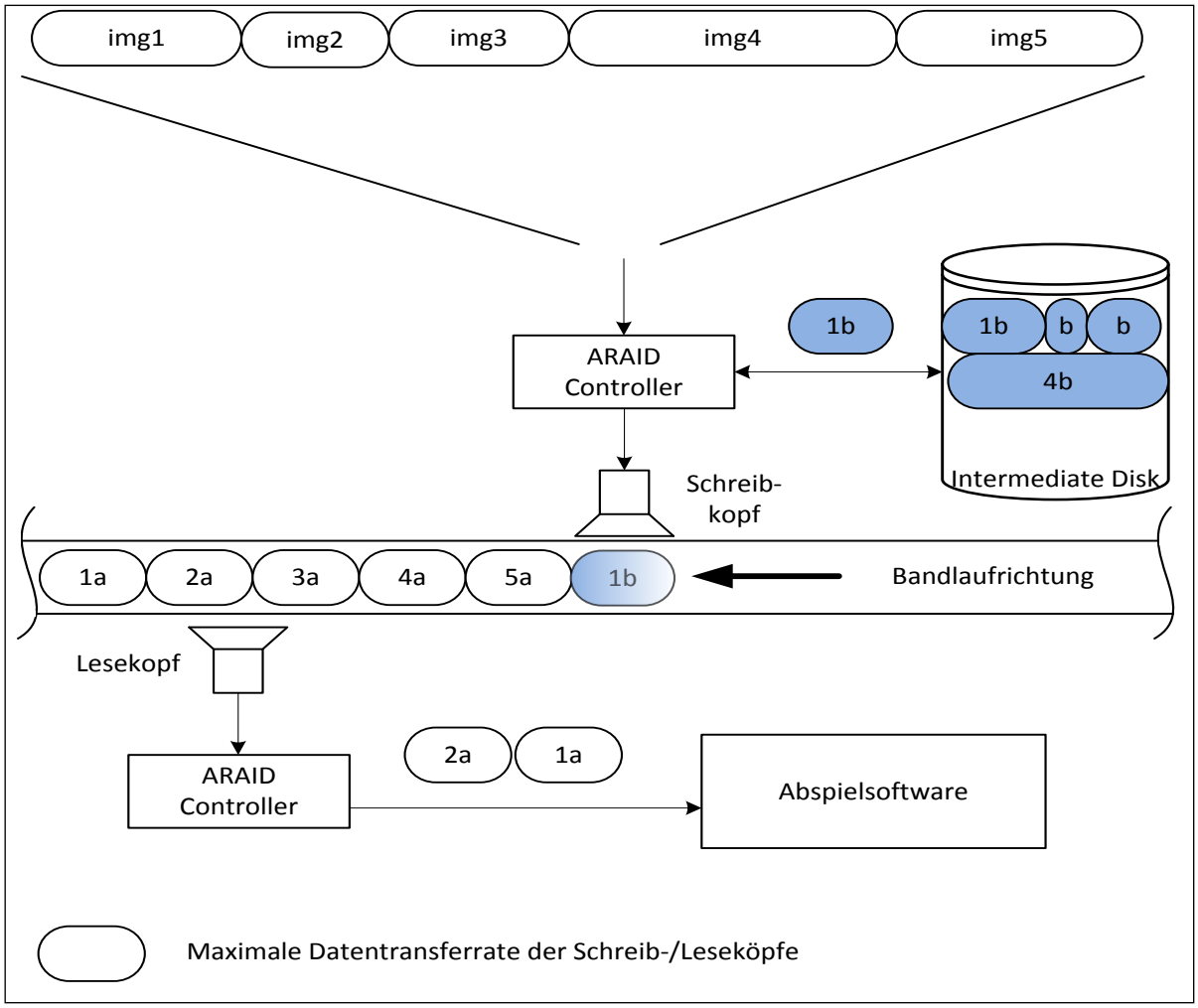

Abbildung 35: Anwendung der ARAID-Technologie in Verbindung mit einem Bandlaufwerk als Speichermedium, Detailbeschreibung im Fließtext

einem separaten Laufwerk zwischengespeichert. Nachdem die echtzeitfähigen Anteile aller Bilder der Sequenz nacheinander auf dem Band gespeichert sind, werden nachfolgend die zwischengespeicherten Datenpakete von der *Intermediate-Disk* gelesen und ebenfalls auf das Band kopiert. Hierdurch sind die echtzeitfähigen Anteile der Eingangsbilder derart auf dem Band gespeichert, dass die Transportgeschwindigkeit der Bandmaschine ausreicht, um die Daten für eine Echtzeitausspielung auszulesen. Nachdem der Lesekopf die Daten an den ARAID-Controller weitergeleitet hat, ist dieser durch die enthaltenen Header-Informationen in der Lage, die Struktur der originalen Datei nachzubilden und fehlende Daten mittels der Substitutionsmethode (vgl. Kapitel 3) zu simulieren. Das Ergebnis kann an die Abspielsoftware weitergeleitet werden.

Es wird deutlich, dass der ARAID-Controller einen Systemkatalog führen muss, um die verbleibenden Längen der nicht-echtzeitfähigen Datenbereiche auffinden zu können. Dennoch bietet dieses Verfahren den Vorteil, dass die Daten auf einem Datenträger zusammen

gespeichert sind und eine Echtzeitvorschau ohne vorheriges und zeitaufwendiges Umkopieren möglich ist.

4.2.4 Implementierung

Die vorgestellten ARAID-Konfigurationen wurden in einem C++-Programm implementiert, um das erwartete Verhalten testen zu können. Das resultierende Programm erlaubt dabei, mehrere Datenträger sowohl in einer RAID-0, wie auch der ARAID-Variante, anzuordnen und Daten entsprechend der gewählten RAID-Methode lesen bzw. schreiben zu können. Als Testsequenz wurden 480 JPEG 2000-Bilder in 4K-Auflösung erstellt. Die Bilder zeigen eine generische Bildsequenz mit großen Detailbereichen. Die maximale Bitrate bei der Codierung wurde mit 250 MBit/s gewählt, was der aktuellen maximalen Datenrate für das Digitale Kino entspricht. Bei den Tests wurden zwei USB-Sticks verwendet, deren Lesegeschwindigkeit vor dem Beschreiben mit jeweils 110 MBit/s ermittelt wurde. Die Wiedergabesoftware läuft auf einer echtzeitfähigen Hardware und fragt entsprechend 24 Bilder pro Sekunde vom (A)RAID-System an, decodiert die Bilder und zeigt sie am angeschlossenen Monitor an (vgl. Abbildung 36). Echtzeitfähigkeit bedeutet in diesem Kontext, dass alle Komponenten der Hardware derart dimensioniert sind, dass eine Wiedergabe ohne Bildaussetzer in der vollen Auflösung und Qualität der vorhandenen Bildsequenz möglich ist. Das Abspielsystem erstellt Statistiken über den Datendurchsatz des Quelldatenträgers sowie der aktuellen Decodierungsleistung. Die verwendete Wiedergabesoftware ist derart implementiert, dass sie so lange das zuletzt erfolgreich decodierte Bild anzeigt, bis ein neues Bild unter Echtzeitbedingung geliefert und decodiert werden kann. Jedes erfolgreich gelieferte Bild wird für die spätere Messauswertung auf der lokalen Festplatte gespeichert.

Nachdem die Sequenz ausgespielt wurde, werden *n* Bilder vom (A)RAID-Speicher (IS_0, IS_1, ..., IS_{n-1}) mit *m* Bildern ($m \leq n$) auf der lokalen Festplatte des Abspielsystems (ID_0, ID_1, ..., ID_{m-1}) verglichen. Wurden z. B. zwei Bilder (IS_1, IS_2) nicht schnell genug an das Abspielsystem geliefert, finden sich keine entsprechenden Versionen (ID_1, ID_2) auf der lokalen Festplatte. Der Vergleich der Bilder erfolgt durch eine *Peak-Signal to Noise-Ratio* (PSNR)-Berechnung nach (4.2), die im geschilderten Szenario die Tupel {IS_0, ID_0}, {IS_1, ID_0}, {IS_2, ID_0}, {IS_3, ID_3} sowie {IS_4, ID_4} vergleicht.

Für die Berechnung des PSNR-Wertes wird zunächst die sog. *Mittlere Quadratische Abweichung* (MQA) zwischen zwei Bildern bestimmt. Für IS_n und ID_n, beide mit der räumlichen Auflösung von $m \times n$, kann der MQA wie folgt berechnet werden:

$$MQA = \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [IS(i,j) - ID(i,j)]^2 \qquad (4.1)$$

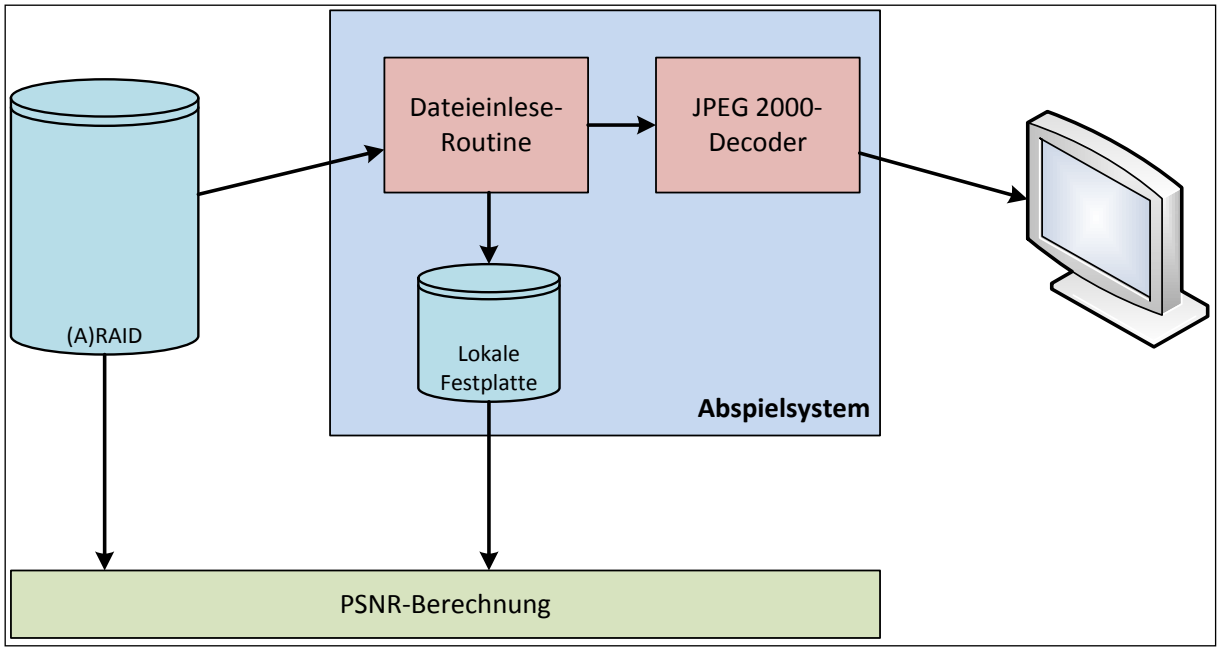

Abbildung 36: Systemübersicht für den Messaufbau der (A)-RAID-Methoden

Der PSNR-Wert berechnet sich nach:

$$PSNR = 10 \cdot \log\left(\frac{MAX_I^2}{MQA}\right) \tag{4.2}$$

MAX_I nimmt dabei den maximal möglichen Wert eines Pixelwertes an. Bei einem Bild mit einer Auflösung von 8 Bit pro Pixel, würde MAX_I den Wert 255 annehmen. Je höher der PSNR-Wert, desto geringer sind die Unterschiede zwischen den verglichenen Bildern.

Besonders beim RAID-0-Verfahren liegen auf der lokalen Festplatte des Abspielsystems und auf dem RAID-System identische Bilder — die Berechnung des PSNR-Wertes ergibt in diesem Fall den Wert ∞. Zur Visualisierung dieser Werte wurde der PSNR-Wert bei identischen Bildern auf Quell- und Ziellaufwerk mit einem festen Wert von 48 dB ersetzt. Die Quellbilder zeigen eine generische Szene, wodurch aufeinanderfolgende Bilder eine große Ähnlichkeit aufweisen. Konnte im RAID-0 Verfahren ein Bild nicht in Echtzeit geliefert werden, fällt der PSNR-Wert nicht auf 0, da das zuletzt erfolgreich decodierte Bild noch eine signifikante Ähnlichkeit zum aktuellen Bild aufweist. Abbildung 37 zeigt die ermittelten Ergebnisse.

Ein PSNR-Wert von 48 dB zeigt dabei ein erfolgreich geliefertes und decodiertes Bild. Jede Abweichung von 48 dB zeigt, dass das Bild nicht in Echtzeit geliefert werden konnte. Bei der Nutzung eines Datenträgers ohne RAID-Verbund (vgl. Abbildung 37-1, oben links) zeigt sich deutlich, dass ein Datenträger nicht in der Lage ist, die gewünschten Bilder in Echtzeit zu liefern. Bei der gewählten Datenrate von 250 MBit/s und einem gemessenen Datendurchsatz von 110 MBit/s ist dieser Effekt zu erwarten. Der Betrachter sieht viele Bildruckler, was eine Vorschau auf das Bildmaterial erschwert. Allerdings werden die erfolgreich gelieferten Bilder

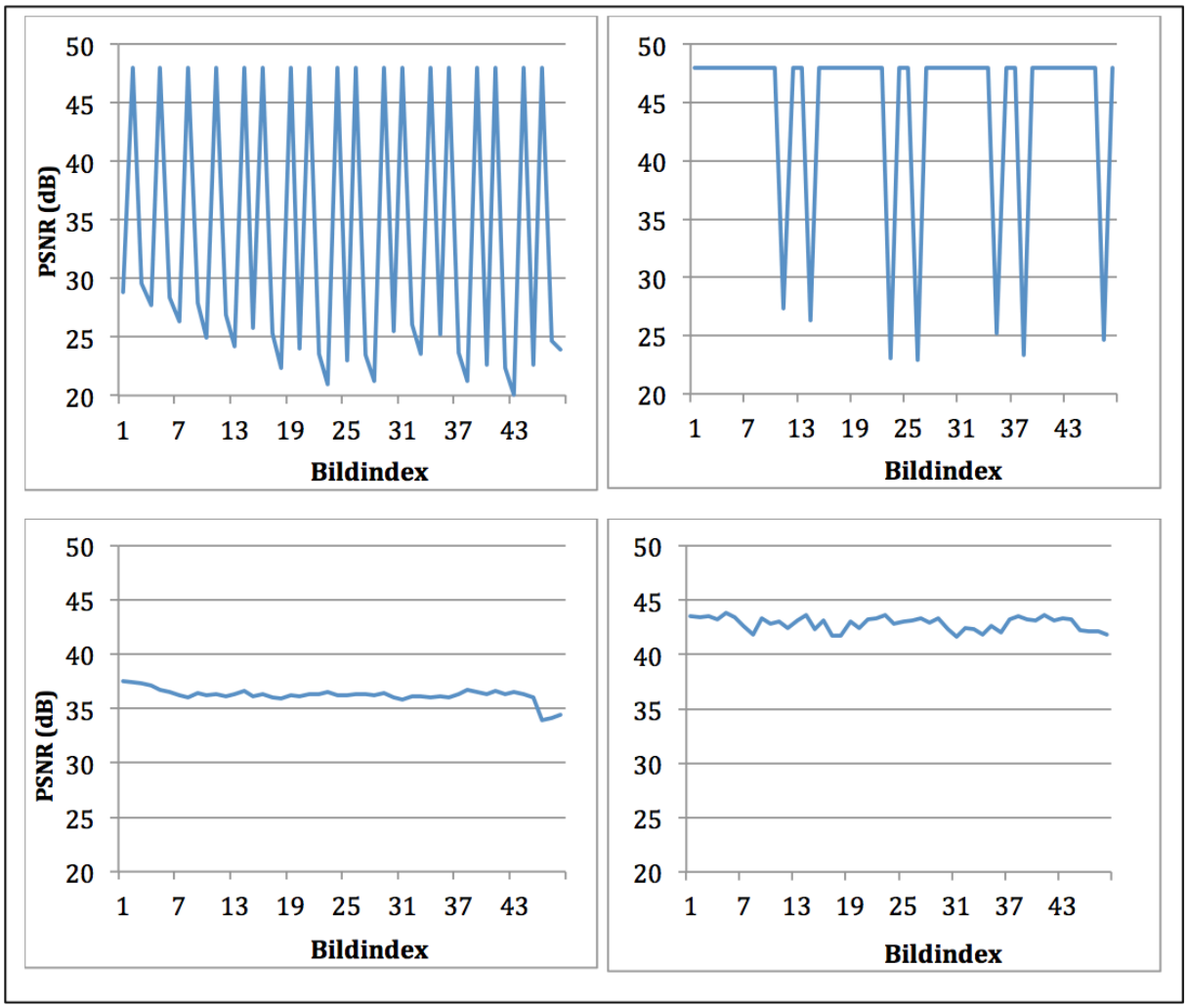

Abbildung 37: PSNR-Vergleich unter Verwendung verschiedener Speichertechnologien für skalierbare Medien: (1) Einfaches Auslesen von einem Datenträger (oben-links), (2) RAID-0-Verfahren mit zwei Datenträgern (oben-rechts), (3) ARAID-Verfahren mit einem Datenträger (unten-links) und (4) ARAID-Verfahren mit zwei Datenträgern (unten-rechts)

in maximaler Qualität angezeigt. Weiter zeigt sich, dass der PSNR-Wert nicht direkt auf 0 dB fällt, wenn ein Bild der Sequenz nicht unter Echtzeitbedingung geliefert werden kann. Dies ist auf die Wahl einer generischen Sequenz zurückzuführen. Das typische RAID-0-Verhalten erhöht den Gesamtdurchsatz des verwendeten Speichers, wie in Abbildung 37-2 (oben rechts) zu erkennen ist. Die Anzahl der Bildaussetzer ist geringer und — analog zur ersten Messung — es liegen alle erfolgreich gelieferten Bilder in der maximalen Qualität vor. Ein anderes Verhalten zeigen dagegen die ARAID-Messungen: Abbildung 37-3 (unten links) zeigt das Ergebnis des adaptiven Schreibvorgangs mit nur einem USB-Stick, der bereits für die Messung ohne RAID-Verfahren genutzt wurde. Es ist deutlich zu erkennen, dass alle Bilder unter Echtzeitbedingung gelesen werden konnten — für den Betrachter kommt es also während des Abspielens nicht zu Bildaussetzern.

Weiter wird deutlich, dass der PSNR-Wert in dieser Messung nie den maximalen Wert erreicht, der Betrachter somit immer ein qualitativ reduziertes Bild erhält. Vorteilhaft ist allerdings, dass bereits der erste Datenträger mit einer Datenrate von 110 MBit/s ausreicht, um die Daten in einer reduzierten Form in Echtzeit ausspielen zu können. Eine weitere Verbesserung der Bildqualität kann durch Verwendung eines zweiten Datenträgers im ARAID-Verbund erreicht werden (vgl. Abbildung 37-4, unten rechts). Auch in dieser Konfiguration kann kein Bild der verwendeten Sequenz in der maximalen Qualität von 48 dB unter den gegebenen Bedingungen vom Datenspeicher gelesen werden, allerdings steigert sich die Qualität — im Vergleich zur Messung in Abbildung 37-3 — noch einmal um etwa 6 dB. Weiter ist auch hier ein Vorteil gegenüber der RAID-0-Methode zu erkennen (Abbildung 37-2), bei der es mit identischen Datenträgern noch häufige Aussetzer während des Abspielens gab.

4.2.5 Bewertung

Die Ergebnisse bestätigen das erwartete Verhalten der vorgeschlagenen ARAID-Verfahren, die neue Möglichkeiten für die Speicherung von skalierbaren Mediendaten ermöglichen. Besonders unter dem Aspekt der Echtzeitausspielung liefert die ARAID-Methode deutlich bessere Ergebnisse als heute verwendete RAID-Konfigurationen. Darüber hinaus kann bereits der erste Datenträger dazu genutzt werden, eine Bildsequenz (bzw. ein Video) in einer reduzierten Variante wiederzugeben, ohne dass die übrigen Datenträger vorhanden sind. Besonders die Messung der Lesegeschwindigkeit der Datenträger vor dem Beschreiben — in Verbindung mit der ARAID-Methode — sichert die Echtzeitfähigkeit beim späteren Ausspielen. Hierdurch entfallen zeitaufwendige Kopierprozesse auf schnellere Datenträger, wodurch viele Arbeitsprozesse innerhalb der Filmdistribution beschleunigt werden können.

Eine weitere Anwendungsmöglichkeit der adaptiven RAID-Verfahren bieten Speichersysteme mit sequenziellem Zugriffsverhalten. Auch hier kann eine Echtzeitfunktionalität der verwendeten Datenspeicher erreicht werden, was besonders im Bereich der Filmarchivierung die Arbeitsprozesse potenziell vereinfachen kann, da auch hier zeitaufwendige Umkopierprozesse von Bandlaufwerken auf schnellere Massenspeicher mit wahlfreiem Zugriff für eine einfache Vorschau auf das Bildmaterial unnötig werden.

Ähnliche Ansätze können für Speichersysteme mit sequenziellem Zugriff (vgl. Abschnitt 2.2.2) entwickelt werden. Hier ist die Idee, lediglich so viele Daten einer skalierbaren Datei sequenziell auf den Datenträger zu speichern, wie dieser später in Echtzeit lesen kann, ohne den Schreib-/Lesekopf erneut re-positionieren zu müssen. Die übrigen Daten, die nicht in das vorberechnete Echtzeitfenster passen, werden nach der Sequenz separat abgelegt. Hierdurch werden die Bilder beim Abspielen zwar in einer reduzierten Qualität und/oder Auflösung abgespielt, können aber ohne vorheriges Umkopieren in Echtzeit betrachtet werden.

5. Verbesserung existierender Verarbeitungsabläufe unter Ausnutzung der Skalierbarkeit

In diesem Kapitel werden drei Bereiche untersucht, bei denen die Nutzung der Skalierbarkeit im adressierten Einsatzgebiet die Echtzeitausspielung ermöglicht, auch wenn die verwendeten Komponenten eigentlich nicht in der Lage sind, die Daten in Echtzeit zu liefern.

5.1 Echtzeitfähige Einlese-Strategie

[17]Massenspeicher benötigen eine gewisse Zeit, um angefragte Daten bzw. eine angefragte Datei zu liefern. Je nach Bauform des Speichers (vgl. 2.2.2) variiert dessen Antwortzeit. Für die Echtzeitwiedergabe (vgl. Kapitel 1) von Videos bzw. einer Sequenz von Einzelbildern ist eine maximale Antwortzeit des Massenspeichers von entscheidender Bedeutung. Sollen z. B. 24 Einzelbilder pro Sekunde abgespielt werden, muss jedes Einzelbild in max. 1/24s an den Decoder geliefert werden.

Heutige SSD-Festplatten sind zwar in der Lage, die Daten in der gewünschten Geschwindigkeit zu liefern, allerdings werden für die Distribution von Mediendaten üblicherweise kostengünstigere und damit oft auch langsamere Festplattentechnologien verwendet. Darüber hinaus werden die Festplatten häufig aus Kostengründen über Schnittstellen mit dem Computersystem verbunden, deren Bandbreite ebenfalls zu gering für die Ausspielung der Bilddaten in Echtzeit ist.

Das Lesen einer generischen Testsequenz mit 480 JPEG 2000-Bildern wurde mit verschiedenen Speichertechnologien getestet. Dabei wurde die maximale Qualität und Auflösung der Bilder abgefragt. Tabelle 9 zeigt, dass der tatsächliche Durchsatz stark variiert und weit unter den Maximalangaben der Hersteller liegt. Während den Tests wurden beide USB-Geräte über eine USB 2.0-Schnittstelle mit dem Computer verbunden. Auch hier zeigt sich, dass die effektive Lesegeschwindigkeit deutlich unter den theoretischen Möglichkeiten der USB 2.0-Schnittstelle (40 MB/s, vgl. Tabelle 4) liegt. Einzig die verwendete SSD-Festplatte wäre in der Lage, eine Dateisequenz — mit einer Datenrate von bis zu 500 MBit/s encodiert wurde — unter Echtzeitbedingung zu liefern, wenn sie direkt im Computersystem verbaut oder durch eine schnellere Schnittstelle als USB 2.0 mit dem Hostsystem verbunden ist.

---

[17] Teilaspekte dieses Abschnitts wurden in [108] veröffentlicht.

Tabelle 9: Geschwindigkeitsmessung verschiedener Speichertechnologien (lesend), Dateigröße: 2 MB, Blockgröße: 8 KB, Zugriffsart: WIN 32 API-uncached, CPU: 3 GHz, Software: PerformanceTest 7.0 [101]

Typ	**Min [MB/s]**	**Max [MB/s]**
Konventionelle Festplatte (HDD)	48	66
Konventioneller-USB-Stick	6	16
USB-Festplatte	8	19
Solid State Drive (SSD)	125	177

Bei der Verwendung nicht skalierbarer Medien stehen nach Ablauf der Zeit lediglich zwei Möglichkeiten zur Verfügung: Das Einzelbild einer Videosequenz kann geliefert und vom Decoder weiter verarbeitet werden oder der Wiedergabeprozess muss das aktuelle Bild überspringen, weil es nicht rechtzeitig vom Massenspeicher geliefert werden konnte — es kommt zu Bildaussetzern während der Wiedergabe. Werden jedoch skalierbare Bilder verwendet, kann das Einlesen von einem Massenspeicher unterbrochen werden, wenn bspw. das Dateisystem feststellt, dass die Zeit nicht ausreicht, um das komplette Bild in der vorgegebenen Zeit zu lesen und zu übertragen. In der Annahme, dass die vorgegebene Zeit ausreicht, um eine Variante mit reduzierter Auflösung und/oder reduzierter Qualität vom Massenspeicher zu lesen, steht nach 1/24s ein kompatibler Datenstrom zur Verfügung, der durch den Decoder verarbeitet und somit Echtzeit erreicht werden kann.

5.1.1 Vorarbeiten auf dem Gebiet der Nutzung skalierbarer Medien

Nach heutigem Kenntnisstand existieren keine Verfahren zum adaptiven Einlesen skalierbarer Medien von Massenspeichern. Vielmehr finden sich Vorarbeiten in anderen Anwendungsgebieten für skalierbare Medien, vor allem im Bereich der Übertragung von skalierbaren Einzelbildern über Datennetze. Ausschlaggebend ist hier, dass die Konzepte heutiger Netzwerke meist nicht geeignet sind, um skalierbare Medien sinnvoll zu übertragen. Viele Technologien, wie bspw. das Caching, basieren im Internet auf sog. *Web-Objects*. Hierbei handelt es sich üblicherweise um eine Datei, die durch eine statische URL angesprochen wird. Dieses Konzept lässt sich nur bedingt für skalierbare Medien nutzen, da sich der gesendete Inhalt ändern kann, obwohl die URL für das Bild identisch bleibt. Zur Umgehung dieses Problems existieren mehrere Lösungsansätze [16,17,43,56,60,84,112], die im Folgenden kurz erläutert werden.

In [56] wird mittels der Metadaten der originalen JPEG 2000-Datei beim Server eine virtuelle Datei (*Vfile*) mit identischer Struktur beim Empfänger erstellt, die keine Bildinformationen enthält. Unterstützt durch ein proprietäres Browser-Plug-In kann ein Klient spezielle Teile der skalierbaren Datei anfragen. Die unterliegende Vfile-API fragt entsprechende Datenpakete an, die nach Erhalt auf Empfängerseite gecacht werden. Serverseitig werden Datenpakete priorisiert versendet, so dass niedrige Auflösungen bzw. niedrige Qualitätsschichten zuerst

beim Empfänger eintreffen. In [16,17] werden serverseitig Vorschaubilder für die JPEG 2000-Dateien (sog. *Proxies*) in einem weit verbreitetem Format, z. B. JPEG oder GIF, erzeugt, damit diese ohne spezielle Zusatz-Software in jedem Webbrowser angezeigt werden können. Eine Indexdatei (ähnlich Vfile) wird durch das Auslesen der *Main*- und *Packet Length Marker* (PLM) erzeugt, allerdings serverseitig abgelegt und mit dem Proxybild verlinkt. Wählt der Nutzer ein Proxy-Bild im Browser aus, lädt eine Zusatz-Software die verlinkte Indexdatei, sowie das JPEG 2000-Bild in geringster Auflösung und Qualität. Mit Hilfe der Indexdatei — die die Struktur der originalen JPEG 2000-Datei abbildet — und dem HTTP-Mechanismus der *byte-ranges* [27] können anschließend — je nach Nutzereingaben — entsprechende Pakete nachgelesen und decodiert werden. Auch in [112] wird der Umstand untersucht, dass JPEG 2000-Bilder nach Part 1 und Part 2 des Standards lediglich durch optionale Marker strukturiert werden können.

Als alternativen Lösungsansatz zu [16,17,56] stellt Taubman ein Verfahren vor, das ohne separate Indexdatei auskommt, serverseitig abstrakte Anfragen der Klienten interpretiert und ermittelt, welche Region in einem Bild angefragt wurde. Hierbei fragen die Klienten keine konkreten Datenpakete, sondern ein sog. *Window of Interest* (*WoI*) in einer bestimmten Qualität und Auflösung an. Um sicherzustellen, dass nur angefragte Daten versendet werden, liest ein Programm die entsprechenden Code-Blöcke von der originalen Datei ein und setzt diese zu einem neuen, standardkonformen JPEG 2000-Bild zusammen. Darüber hinaus hält der Server optional ein Abbild des Klienten-Caches, um dadurch zu vermeiden, dass Daten mehrfach versendet werden. Im Vergleich zu den Verfahren [16,17,56] wird deutlich, dass zu Beginn einer Session keine Indexdatei ausgetauscht werden muss. Weiter kann der Server entscheiden, wie viele Daten bei einer Anfrage zurückgeliefert werden. So können die Antworten für eine Anfrage — in Abhängigkeit der aktuellen Auslastung des Servers — unterschiedlich ausfallen. Nachteilig wirkt sich jedoch aus, dass eine spezielle Implementierung eines Servers nötig ist und keine Standard HTTP-Server genutzt werden können. Große Teile dieser Arbeit wurden anschließend in den JPEG 2000-Standard Part 9 [43] übernommen. Eine Komplettintegration in einen Webbrowser anstatt zusätzlicher Software beim Klienten wird in [84] vorgestellt. Der Server liefert hier Teilbilder entsprechender JPEG 2000-Originaldateien in Abhängigkeit der Nutzeranfrage. A-priori existieren die drei Auflösungsstufen, die vom Klienten angefragt werden können: (i) 64x64 (Vorschau), (ii) 256x256 (Medium) und (iii) die volle Auflösung laut Main-Header der originalen Datei. Bei der ersten Anfrage eines Bildes sendet der Server den Main-Header und entsprechende Bilddaten für die angefragte Auflösung, bei weiteren Anfragen werden lediglich Bilddaten und Anzahl der Auflösungsstufen gesendet. Empfängerseitig werden die empfangenen Daten zusammengeführt und anschließend decodiert, bevor das Bild im Webbrowser angezeigt werden kann. Durch die vorgegebenen Auflösungsstufen zeigt sich das Verfahren nicht so flexibel wie die bereits vorgestellten Verfahren, bietet aber Vorteile durch eine Komplettintegration im Webbrowser.

In [60] wird das Konzept *Vmedia* (virtual media access protocol) vorgestellt, das auf den Verfahren von [16,17] aufsetzt. Hierbei handelt es sich um ein proprietäres Verfahren, das den entfernten Zugriff auf Mediendaten sowie die Verwaltung eines Caches beim Klienten definiert. Hierzu nutzt das Verfahren u. a. eine sog. *Companion-Datei*, die der Index-Datei im Ansatz von Deshpande et al. entspricht. Auf Anfrage des Nutzers werden sukzessiv Teile der Datei nachgeladen. Hierbei wählen Li und Sun eine Strategie, wonach Qualitätspakete priorisiert an die aufrufende Applikation gesendet werden. Hierdurch soll erreicht werden, dass der Nutzer innerhalb kürzester Zeit seine nächste Anfrage bestimmen kann.

Zum einen können die geschilderten Verfahren teilweise für die Entwicklung einer echtzeitfähigen Einlesestrategie von potenziell langsamen Massenspeichern übernommen werden. Zum anderen existiert bereits heute die Möglichkeit, die Durchsatzleistung von langsamen Massenspeichern zu kompensieren, indem bspw. der Benutzer manuell die Auflösung in der Abspiel-Software reduziert. Die Software fordert dadurch eine reduzierte Variante von der Festplatte an und der Prozess kann ggf. beschleunigt werden. Entsprechende Einstelloptionen in der Abspiellösung sind hierfür nötig, was vor allem bei Hardware (a) nicht immer gewährleistet ist und (b) nicht durch ein einfaches Software-Update erreicht werden kann. Weiter muss der Einstellungsprozess für jedes System — in Abhängigkeit von dem verwendeten Speicher und der Dateigröße der Bilder — erneut justiert werden. Deshalb wurde im Rahmen dieser Arbeit ein adaptives System zum Einlesen unter Echtzeitbedingungen entwickelt, das im Folgenden vorgestellt wird.

5.1.2 Entwicklung eines echtzeitfähigen Dateisystems für skalierbare Medien am Beispiel von JPEG 2000

Die vorgestellten Verfahren [56,60] beruhen auf einer Client-/Server-Architektur, bei der die Dateigröße — je nach gelieferter Version — auf der Empfängerseite angepasst wurde. Dieses Vorgehen ist sinnvoll, denn die zu übertragende Datenmenge über ein potenziell langsames Netzwerk sollte minimiert werden. Innerhalb eines Computersystems zeigen sich jedoch andere Engpässe beim Abspielen von Einzelbild- oder Videoströme mit hohen Datenraten. Abbildung 38 zeigt dabei die prinzipielle Prozesskette: Ein Prozess, der üblicherweise im Benutzer-Modus läuft, öffnet eine Datei, indem er Zugriff vom Betriebssystem anfordert. Nach Weiterleitung der Anfrage an das Dateisystem prüft dieses die Nutzerberechtigungen und gewährt dem Prozess bei Erfolg den gewünschten Zugriff auf die Datei, indem es ein gültiges *Handle* liefert. Nachfolgende Schreib-/Leseanfragen werden über das Betriebssystem an das Dateisystem weitergeleitet. Je nach adressiertem Massenspeicher wird ein entsprechender Treiber angesprochen, der den internen Aufbau des eigentlichen Speichers (Festplatte, DVD, Diskette) abstrahiert. Der Engpass tritt in dieser Architektur üblicherweise zwischen dem Massenspeicher und dem entsprechenden Treiber auf, da die verfügbaren Speicher (vgl. 2.2.2) einen signifikant geringeren Datendurchsatz als der Rest der

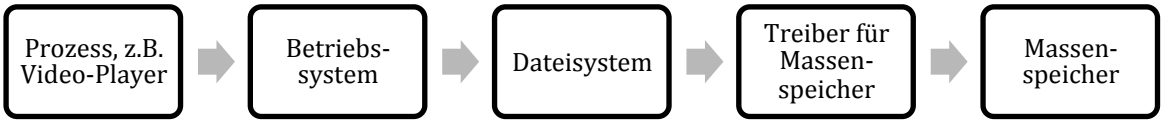

Abbildung 38: Prozesskette beim Zugriff auf einen Massenspeicher, initiiert durch einen Benutzer-Prozess

Prozesskette bieten. Die bekannten, auf einer Client-/Server-Architektur basierenden Lösungsansätze, müssen für die Anwendung innerhalb einer Computerarchitektur zur Videobearbeitung um die folgenden Aspekte erweitert werden:

1. Eine Änderung der Dateigröße in Abhängigkeit der gelieferten Daten ist nicht möglich. Listing 4 zeigt hierzu das übliche Vorgehen beim Öffnen und Auslesen einer Datei vom Datenträger. Entscheidend dabei ist, dass mittels dem Aufruf *stat* (Zeile 3) am Dateisystem angefragt wird, wie groß die Datei ist. In Abhängigkeit dieser Größe wird anschließend Speicher allokiert und von der Quelldatei gelesen. Eine spontane Änderung der vorher ermittelten Dateigröße, durch Anwendung der bekannten Verfahren zur Nutzung der Skalierbarkeit, hätte zur Folge, dass die Applikation nicht vorhandene Datenbereiche anfordert (Zeile 7).

2. Eine Ausspielung muss üblicherweise in Echtzeit erfolgen. Dabei sollen so viele Daten wie möglich gelesen und decodiert werden. Dieser Prozess soll automatisiert erfolgen.

Auf Grund der Eigenschaften innerhalb einer Computerarchitektur — in Kombination mit den erweiterten Anforderungen der Echtzeitausspielung — wird hier ein neues Konzept vorgestellt, das auf der Annahme beruht, dass zwar nicht die komplette Datei vom Massenspeicher unter Echtzeitbedingung gelesen werden kann, jedoch aber ein signifikanter Teil der skalierbaren Daten im vorgegebenen Zeitfenster auslesbar ist. Um die Daten in Echtzeit liefern zu können, wird deshalb die Lese-Routine innerhalb eines virtuellen Dateisystems um einen Zeitbegrenzer (engl. *Timer*) erweitert. Dieser Timer stellt sicher, dass die aufrufende Applikation (a) innerhalb einer vorher festgelegten Zeit eine Antwort enthält und dass (b) die Dateistruktur der gelieferten Daten nach Ablauf der Zeit der Struktur der originalen Datei entspricht. Die zur Verfügung stehende Zeit kann dabei auf verschiedene Weise übergeben werden. In der aktuellen Implementierung ist es ein Parameter, der beim Start des Dateisystems mit 1/24 Sekunden festgelegt ist, um dadurch eine Echtzeitausspielung von 24 Bildern pro Sekunde gewährleisten zu können. Fehlende Daten, die nach Ablauf der gesetzten Zeit noch nicht vom Datenträger eingelesen wurden, kompensiert die Substitutionsmethode (vgl. Kapitel 3).

Abbildung 39 zeigt dabei das prinzipielle Vorgehen: Wird eine Datei vom (virtuellen) Dateisystem angefragt, wird zunächst geprüft, ob es sich um eine skalierbare Datei handelt. Nichtskalierbare Dateianfragen werden ohne weitere Bearbeitung an das reale Dateisystem

```
01:   Algorithm void main (int argc, char **argv)
02:     struct stat file_status;
03:     stat(argv[0], &file_status)
04:     FILE *fp = fopen (FILENAME, "r");
05:     char *buffer = NULL;
06:     buffer = (char *) malloc(file_status.st_size+1);
07:     fread(buffer, file_status.st_size,1,fp);
08:     buffer[file_status.st_size] = '\0';
        /* Do something with the buffer*/
09:     fclose(fp);
10:     free(buffer);
11:   return 0;
```

Listing 4: Pseudo-Code-Darstellung zum Öffnen und Lesen einer Quelldatei ohne Fehlerprüfung

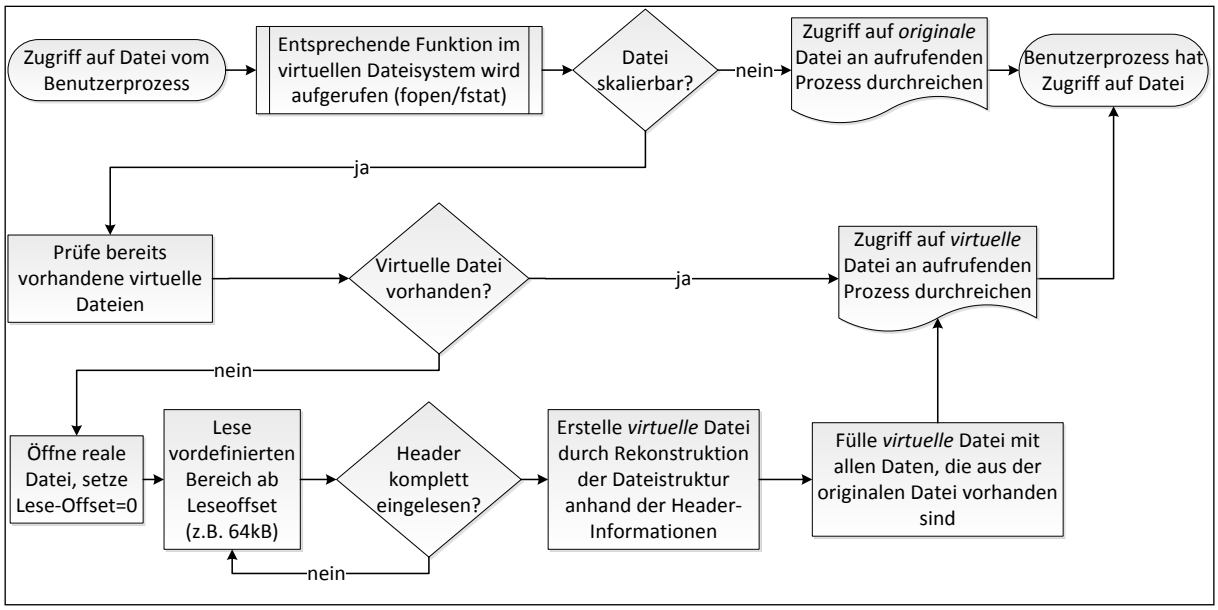

Abbildung 39: Programmablauf beim ersten Zugriff auf eine Datei über das virtuelle Dateisystem

weiter gegeben und der Prozess ist beendet. Handelt es sich um eine skalierbare Datei, folgt die Prüfung, ob die Datei schon als Variante innerhalb des virtuellen Dateisystems vorliegt. Gibt es bereits eine virtuelle Variante, erhält der aufrufende Prozess durch ein sog. *Datei-Handle* Zugriff auf die Datei. Ansonsten öffnet das virtuelle Dateisystem die reale Datei und liest die ersten Datenblöcke — unter Berücksichtigung der Zeitvorgabe — ein. Die Menge der Daten ist dabei frei wählbar, ein Wert von 64 KB pro Lesezugriff hat sich jedoch als solide bewährt, da hierdurch der Gesamtdurchsatz des Datenträgers nicht merkbar sinkt und häufig entschieden werden kann, ob ein weiterer Zugriff innerhalb des vorgegebenen Zeitfensters möglich ist. Weiter sind in 64 KB üblicherweise alle Header-Daten einer skalierbaren Datei enthalten. Ist das nicht der Fall, wird die Einleseprozedur so lange wiederholt, bis der komplette Header im virtuellen Dateisystem vorhanden ist. Anhand der optionalen PLT-Marker (vgl. auch [113]) kann anschließend — analog zum Konzept VFile (vgl. [56]) — eine virtuelle Datei mit identischer Struktur im Speicher des virtuellen Dateisystems angelegt werden. Sind beim Einlesen der Headerdaten auch schon Bildinformationen des ersten Pakets (oder der ersten Pakete) eingelesen worden, werden diese entsprechend in die virtuelle Datei

kopiert. Weiter speichert das virtuelle Dateisystem intern den Offset, an dem zuletzt in der Datei gelesen wurde. Dieser Wert entspricht bei nachfolgenden Aufrufen dem Lese-Offset in der gewünschten Datei. Zusätzlich wird noch der Ende-Marker (EOC, siehe [113]) an die letzte Position des Puffers im Speicher eingesetzt — die Position ergibt sich aus der Größe der originalen Datei. Abschließend liefert auch dieser Zweig der Prozedur ein Datei-Handle an den aufrufenden Prozess. Allerdings ist es in diesem Fall nicht das Handle auf die originale Datei, sondern auf die virtuelle Datei innerhalb des Dateisystems.

Die Prozedur für nachfolgende Leseaufrufe ist in Abbildung 40 dargestellt: Zunächst wird geprüft, ob die virtuelle Datei vorhanden ist. Ist das nicht der Fall, wird das übergebene Handle als ungültig identifiziert und der aufrufende Prozess erhält eine Fehlermeldung. Im positiven Fall ermittelt das virtuelle Dateisystem den Speicherort der virtuellen Datei. Anhand des internen Offsets kann weiter berechnet werden, ob der angefragte Bereich bereits komplett vom Massenspeicher gelesen wurde. Dies könnte z. B. der Fall sein, wenn der aufrufende Prozess Daten im Bereich von 0-65535 Byte anfragt, da dieser Bereich bereits beim Öffnen der Datei ausgelesen worden ist. In diesem Fall liefert das Dateisystem die angefragten Daten an den aufrufenden Prozess zurück. Werden jedoch Daten angefragt, die noch nicht vom Massenspeicher eingelesen wurden, kommt die echtzeitfähige Einlesestrategie zum Tragen. Hierbei werden zwei Zeiten berücksichtigt. Zum einen der interne Timer t und zum anderen eine Echtzeitvorgabe t_{RT}, die beide dem Dateisystem als Parameter übergeben werden. Bei jedem Zugriff auf die reale Datei wird t mit dem Ziel neu gestartet, den Lesevorgang abzubrechen, bevor $t > t_{RT}$ ist. Anschließend werden so lange weitere Bereiche der originalen Datei eingelesen, bis der Algorithmus berechnet, dass ein weiterer Schleifendurchlauf die Echtzeitbedingung verletzen würde. Bei jedem Durchlauf wird dabei der aktuelle Lese-Offset gespeichert. Zusätzlich wird noch ein Zeitpuffer eingebaut, da auch die nachfolgenden Prozedurschritte innerhalb des virtuellen Dateisystems zusätzliche Zeit in Anspruch nehmen. Steht nicht mehr ausreichend Zeit für einen weiteren Durchlauf zur Verfügung, bricht die Schleife ab. Wie in Kapitel 3 erläutert, werden nur komplette Pakete des skalierbaren Datenstroms an den aufrufenden Prozess zurückgeliefert. Pakete, die nur teilweise eingelesen werden konnten, werden im letzten Schritt der Leseprozedur durch Nullpakete ersetzt. Abschließend werden dem aufrufenden Prozess die Daten für den angefragten Bereich übergeben und die Prozedur beendet.

Durch Einbau der Timer-Funktionalität innerhalb der Leseroutine kann eine berechenbare Antwortzeit des Dateisystems erreicht werden. Um dem Fall vorzubeugen, dass eine weitere Leseiteration deutlich länger als erwartet dauert, weil es bspw. zu konkurrierenden Zugriffen auf einem Datenträger kommt, wird innerhalb des Systems ein zweiter Thread gestartet, der die Leseanfrage abbrechen kann, wenn die maximale Ausführungszeit erreicht ist. Für den aufrufenden Prozess ist das Vorgehen vollkommen abstrakt und das Ergebnis ist in jedem Fall eine valide JPEG 2000-Datei. Es kommt weder zu Änderungen der Dateigröße noch der

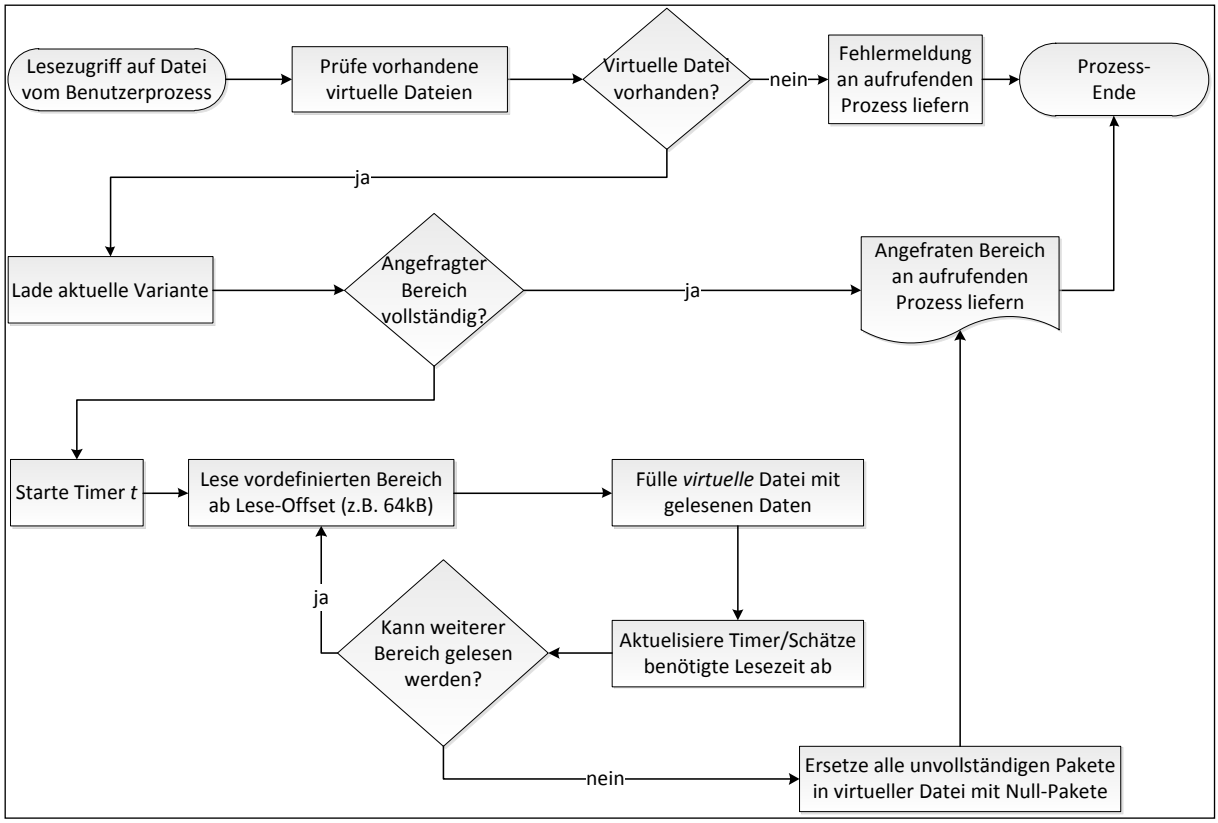

Abbildung 40: Programmablauf für den lesenden Zugriff über das virtuelle Dateisystem

internen Struktur, so dass eine Abspiel- oder Decodierungs-Software nicht auf solche spontanen Änderungen reagieren muss.

Evaluation:

Zur Evaluierung der beschriebenen Verfahren wurden zwei JPEG 2000-Testsequenzen mit je 500 Bildern und einer maximalen Auflösung von 4K verwendet. Weiter beinhalteten beide Testsequenzen Bilder mit fünf Qualitäts- sowie Auflösungsstufen. Neben dem Header beinhaltete somit jedes Bild 90 Datenpakete (5 Qualitätsschichten * (5+1) Auflösungsstufen * 3 Farbkomponenten), die bei Bedarf mit Nullpaketen ersetzt werden konnten. Die durchschnittliche Dateigröße belief sich auf 8 MB. Die Testsequenzen unterscheiden sich lediglich in der Progressionsreihenfolge, die jeweils mit LRCP (qualitätsorientiert) und RLCP (auflösungsorientiert) gewählt wurden.

Als Grundlage für die Entwicklung einer echtzeitfähigen Einleseroutine kam das bereits vorgestellte Dateisystem (vgl. 4.1) zum Einsatz, das durch die Ausführung im Benutzermodus eine geringere Durchsatzleistung im Vergleich zu nativen Dateisystemen liefert. Die Anfrage der Bilder vom Dateisystem wurde durch ein selbstentwickeltes Testprogramm durchgeführt, da hierdurch sichergestellt werden kann, dass pro Leseaufruf nur eine Anfrage an das Dateisystem gestellt wird. Einige Systemprogramme fragen die Daten nicht komplett an, sondern zerteilen die Anfragen ihrerseits in mehrere Subanfragen bestimmter Größe. Da jede Anfrage auf eine Datei im Prototypen des Dateisystems dazu führt, dass der Timer zur

Sicherstellung der Echtzeitbedingung zurück gesetzt wird, würden mehrere Subanfragen pro Leseiteration das Messergebnis verfälschen. Das Testprogramm führt dementsprechend die wesentlichen Prozeduren aus: (1) Ordner innerhalb des Dateisystems für skalierbare Mediendaten öffnen und alle JPEG 2000-Dateien auflisten, (2) Leseanfrage für jedes JPEG 2000-Bild stellen und (3) das Ergebnis auf einem separaten Datenträger speichern. Anhand der abgespeicherten Bilder — die das Dateisystem unter Echtzeitbedingungen zurückliefert — kann im nachfolgenden Schritt die Funktionalität des Dateisystems überprüft werden. Hierzu wurde sowohl die originale Sequenz — wie auch die über das Testprogramm angefragte und abgespeicherte Sequenz — decodiert und der PSNR-Wert zwischen den Varianten ermittelt.

Abbildung 41 und Abbildung 42 zeigen dabei die Resultate der PSNR-Berechnung in Abhängigkeit verschiedener Bildwiederholraten, die innerhalb des virtuellen Dateisystems eingestellt wurden. Die Ausführungszeit des Dateisystems für skalierbare Medien — und die damit verbundenen Kontext-Wechsel des Betriebssystems — erfordern die Einstellung eines großen Zeitpuffers, der vom Dateisystem für Prozeduren neben der eigentlichen Leseanfrage benötigt wird. Durch Tests wurde auf dem verwendeten System ein Wert von 31 ms ermittelt, der nicht für Leseanfragen verwendet werden kann. Dementsprechend wurden für die Auswertungen Bildwiederholraten zwischen 1 fps und 30 fps gewählt. Das Dateisystem lieferte die Daten in allen Testläufen innerhalb der vorgegebenen Zeit. Lediglich bei einer definierten Bildwiederholrate von 1 fps konnten die Bilder komplett vom realen Massenspeicher gelesen und an den aufrufenden Prozess zurückgeliefert werden.

Demnach sind die originalen und vom Dateisystem gelieferten Bilder identisch — ein nachfolgender PSNR-Vergleich würde für alle Vergleiche ∞ ergeben. Zur Visualisierung wurden alle ∞ Werte mit einem frei gewählten Wert von 52 dB ersetzt. Die Messungen zeigen das zu erwartende Verhalten des echtzeitfähigen Dateisystems: Je höher die Bildwiederholrate, desto geringer die verfügbare Zeit zum Lesen vom realen Datenträger, desto geringer der Anteil realer Daten in der virtuellen Datei. Entsprechend ist der PSNR-Wert geringer im Vergleich zu den niedrigeren Bildwiederholraten und der originalen Dateivariante. Weiter ist zu erkennen, dass die gewählte Progressionsreihenfolge der Bildsequenz unter gewissen Randbedingungen einen Einfluss auf die zurückgelieferte Datenmenge hat. So zeigt sich bspw., dass sich bei der Progressionsreihenfolge LRCP und einer Bildwiederholrate von 24 fps, im Indexbereich von etwa 70-220 ein höherer PSNR-Wert im Vergleich zur RLCP-Variante erzielen lässt. Dies lässt sich durch die Struktur der encodierten Bilder erklären, die in diesem Beispiel große Pakete mit Auflösungsdaten und kleinere Pakete mit Qualitätsdaten enthalten. Diese Verteilung ist der Grund dafür, dass die Pakete nicht komplett unter Echtzeitbedingung eingelesen werden können und entsprechend der oben geschilderten Verfahren durch Nullpakete ersetzt werden. Bei der LRCP-Progressionsreihenfolge hingegen werden zunächst viele kleine Pakete eingelesen und weniger Informationen — die schon vom realen Datenträger gelesen wurden — verworfen.

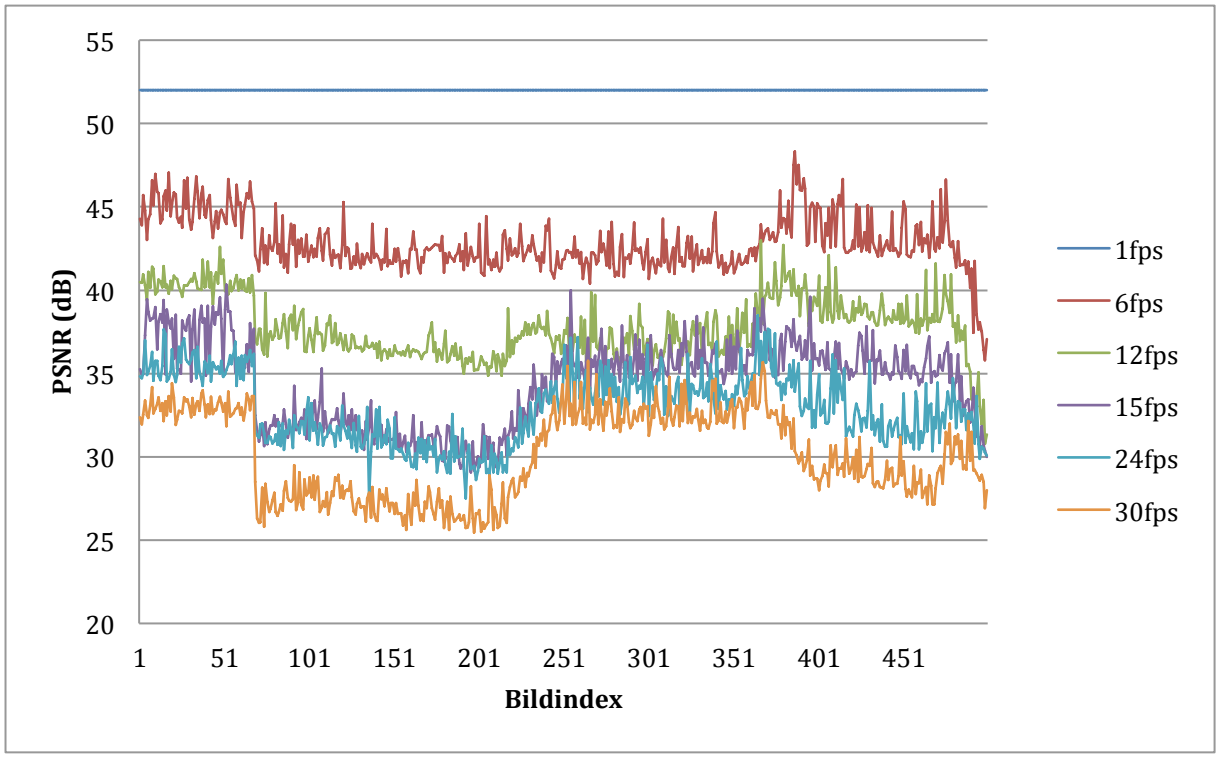

Abbildung 41: PSNR-Vergleich zwischen decodierter Originalsequenz und der unter Echtzeitbedingung gelieferten Variante vom virtuellen Dateisystem mit verschiedenen Bildwiederholraten und der Progressionsreihenfolge LRCP

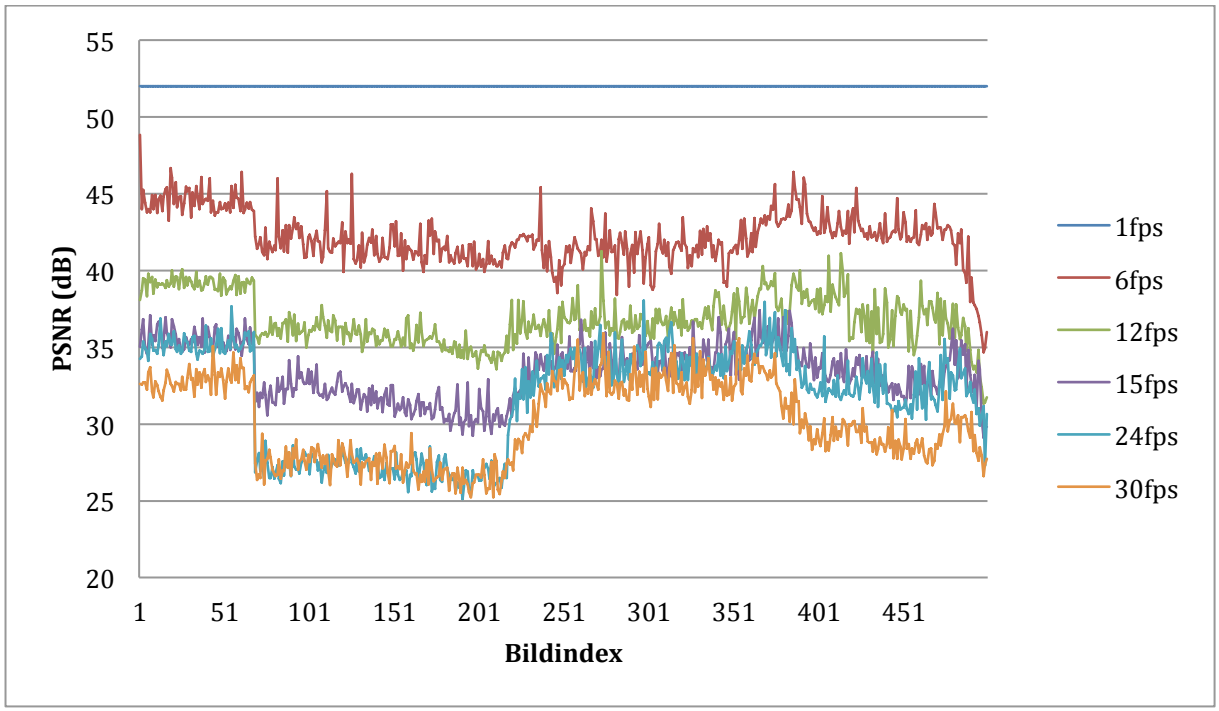

Abbildung 42: PSNR-Vergleich zwischen decodierter Originalsequenz und der unter Echtzeitbedingung gelieferten Variante vom virtuellen Dateisystem mit verschiedenen Bildwiederholraten und der Progressionsreihenfolge RLCP

Bewertung:

Die Implementierung der Verfahren in einem virtuellen Dateisystem bietet den entscheidenden Vorteil, dass die Entwicklung mit einer Hochsprache und abstrahiert von der verwendeten Hardware erfolgen kann. Nachteilig wirkt sich jedoch die Durchsatzleistung des Gesamtsystems aus, die deutlich hinter den Durchsatzraten realer Dateisysteme bleibt. Gründe dafür sind die häufigen Kontextwechsel, sowie die Tatsache, dass eine zusätzliche Kopie der Daten nötig ist, da das virtuelle Dateisystem wie ein weiterer Zwischenschritt in der Verarbeitungskette wirkt. Glücklicherweise erfolgen die Kopieroperationen im Speicher und können somit sehr schnell durchgeführt werden. Eine Portierung der hier geschilderten Verfahren in ein reales Dateisystem, das im Kernel-Modus des Betriebssystems ausgeführt wird, ist demnach eine logische Optimierung, die allerdings nicht im Rahmen dieser Arbeit durchgeführt wurde.

Weiter ist es offensichtlich, dass durch Ersetzung realer Datenpakete durch Nullpakete die Informationsrate — und dadurch die Bildqualität — zum Teil signifikant verringert wird. Dieses Verfahren eignet sich somit potenziell nicht für die finale Vorstellung eines Kinofilms auf einer großen Leinwand, auf der fehlende Bildinformationen besonders schnell auffallen. Allerdings ist die maximale Bildqualität nicht in allen Schritten der Filmpostproduktion und Filmdistribution entscheidend. Häufig ist eine schnelle Vorschaumöglichkeit auf Filmsequenzen unter Echtzeitbedingungen wichtiger als die maximale Qualität. Dies ist z. B. bei der Eingangskontrolle in Kinos der Fall. Hier werden täglich viele Filme auf Festplatten mit langsamen USB-Schnittstellen und/oder DVDs angeliefert. Filmvorführer können durch Sichtprüfung feststellen, ob es sich tatsächlich um den angegebenen Film handelt, ohne die Daten zuvor auf ein schnelleres Speichermedium umkopieren zu müssen. Die hier vorgestellte, echtzeitfähige Einlesemethode erlaubt die Kontrolle der skalierbaren Medien für diese Anwendungsfälle.

5.2 Parametrisierbare Dateiaufrufe

Der Aufruf einer Datei innerhalb der relevanten Dateisysteme erfolgt über einen Identifikator (Uniform Resource Identifier – URI), der eine optionale Pfadangabe sowie den Dateinamen enthält. Versucht ein Prozess eine Datei aufzurufen, prüft das jeweilige Rechtesystem, ob der Prozess diese Operation ausführen darf. Für nicht skalierbare Dateien reicht dieses Konzept weitestgehend aus, da eine Datei nur in ihrer Gesamtheit sinnvolle Daten für einen Prozess liefert. Häufig jedoch werden Mediendaten in verschiedenen Varianten benötigt. In vielen Fällen reicht z. B. eine kleine Variante eines Bildes zu Vorschauzwecken aus. Das gleiche Bild kann aber später — z. B. von einer Bearbeitungssoftware für Fotos — in der vollen Auflösung benötigt werden. Können entsprechende Programme die skalierbaren Daten eigenständig interpretieren, ist das Konzept der Pfadangabe weitestgehend ausreichend. Fehlt einer entsprechenden Software allerdings diese Kenntnis, kann sie keinen Nutzen aus der Skalierbarkeit ziehen.

Aus diesem Grund wird hier das Konzept der parametrisierbaren Dateiaufrufe vorgestellt, bei dem aufrufende Prozesse Anfragen an das Dateisystem mit Parametern versehen, um eine bestimmte Variante des skalierbaren Medientyps zu erhalten. Derartige Verfahren werden beim Aufruf von Webseiten eingesetzt, um z. B. entsprechende Variablen an ein PHP-Skript zu übergeben. Nach heutigem Kenntnisstand existieren noch keine Verfahren im Bereich der Dateisysteme, die ähnliche Konzepte nutzen, um eine alternative Version einer Mediendatei vom Dateisystem anzufragen. Vielmehr finden sich Vorarbeiten auf dem Gebiet der Übertragung von skalierbaren Einzelbildern über Netzwerke. So zeigt [114] die Konzepte des interaktiven JPEG 2000-Protokolls (JPEG 2000 Interactive Protocol - JPIP), das — im Gegensatz zur Verwendung einer Indexdatei [16,17,112] — ein abstrakteres Anfragemodell für das gewünschte WoI nach dem Vorbild aus [112] favorisiert. Hierdurch werden Bildausschnitte nicht direkt angefragt, sondern lediglich Wünsche an einen Bild-Server formuliert. Dieser verfügt über die Möglichkeit, die Anfrage, je nach aktueller Auslastung, zu verändern. Auch hier kann der Klient empfangene Daten lokal speichern, um bei zukünftigen Anfragen bereits erhaltene Informationen wiederzuverwerten. Hierzu muss der Klient allerdings in der Lage sein, JPIP-Anfragen entsprechend seines aktuellen Cache-Zustandes umzuformulieren. Ebenso kann der Server ein Abbild des Klient-Caches halten, um dadurch zu ermitteln, welche Daten ggf. schon beim Empfänger vorhanden sind. Über verschiedene Instruktionen kann der Klient dem Server-Cache-Management Anweisungen senden, damit das Abbild im Server konsistent zum Klienten bleibt, wenn dieser sich z. B. entscheidet, Daten aus seinem lokalen Cache zu löschen. Je nach Anfrage eines Klienten muss der Server neue Datenpakete zusammensetzen, was in den meisten Fällen dazu führt, dass die optimale Anordnung der Datenpakete — die während des Komprimierungsprozesses entsteht — aufgelöst wird. In [115] wird deshalb ein Verfahren vorgestellt, das eine optimale Reihenfolge der Datenpakete, auch nach Neukomposition, gewährleistet, ohne dabei die Datei erneut komprimieren zu müssen. Optimal in diesem Kontext bedeutet, dass an jedem Punkt der Sequenz die Verzerrung im WoI — in Abhängigkeit der versendeten Datenmenge — so klein wie möglich ist. Hierzu wird für jedes Paket des aktuellen WoIs die sog. *Rate-Distortion-Slope (R-D)* berechnet und anschließend alle Pakete eines Bildbereiches, die in einem gewissen R-D-Bereich liegen, in einem Qualitäts-Layer gebündelt und versendet. In [78] wird eine Erweiterung zu JPIP Namens JPIP-Web (JPIP-W) vorgestellt, die mit den Standard-Cache-Verfahren des Internets verwendet werden kann. Hierzu unterteilt JPIP-W bereits encodierte JPEG 2000-Bilder in Blöcke, die mehrere Pakete enthalten und vergibt für jeden Block eine URL. Jeder Block ist dadurch automatisch ein sog. *Web-Object*, eine Einheit, die durch Standard-Web-Proxy-Server zwischengespeichert wird. Eine Anfrage an den JPIP-W Server wird mit einer Liste von URLs beantwortet. Da sich die URL für einen Block — im Gegensatz zu JPIP — nicht ändert, kann ein Proxy-Cache auf die potenziell zwischengespeicherten Daten zurückgreifen. JPIP-W schneidet in der Initialphase schlechter ab als JPIP, zeigt jedoch deutlich bessere Ergebnisse, wenn bereits Daten in den Zwischenspeichern vorhanden sind. Bei der Nutzung des JPIP-Protokolls in einem Proxy-Server [58] identifizie-

ren die Autoren zwei elementare Probleme: (i) Serving Problem: In welcher Abfolge müssen Daten vom Proxy an den anfragenden Klienten gesendet werden? Da der Proxy keine Kenntnis über das Gesamtbild besitzt, sondern lediglich anhand von Metadaten aus einem empfangenen JPIP-Datenpaket erkennen kann, zu welchem Bildbereich die Daten gehören und wie viele Qualitätsschichten existieren, muss der Proxy schätzen, wie groß die Pakete für die Qualitätsschichten sind. Lima et al. stellen hierzu zwei Verfahren vor, die nicht ideal sind, aber sehr gute Ergebnisse liefern. (ii) Aggregation-Problem: Hierbei stellt sich die Frage, wie parallele Anfragen mehrerer Klienten so umformuliert werden können, dass die Qualität der Bilder — in Abhängigkeit von der verfügbaren Bandbreite — über die Gesamtheit aller Anfragen maximal verbessert wird. Das Ziel ist dabei, den Faktor zwischen Entstellung (engl. *Distortion*) und Länge der angefragten Daten zu maximieren.

Die Verfahren von Deshpande und Zeng [16,17] sowie die Mechanismen des JPIP-Protokolls [114] benötigen bei der Kommunikation zwischen Klient und Server zusätzliche Daten. Gibt es bei [16] bzw. [17] nur einen Zugriff, um die Index-Datei auszulesen, können bei JPIP mehrere Runden nötig sein, bevor die ersten Bilddaten ausgetauscht werden können. Die Arbeit von Ortiz et al. [77] schlägt ein weniger flexibles, dafür aber effektiveres Vorgehen mit weniger Indexdaten vor. Unter dem Namen JPEG 2000 Light (J2L) stellen sie ein Verfahren mit einer festen Struktur der JPEG 2000-Bilder vor. Der Main-Header muss die optionalen Tile-Part-Längen beinhalten und die Progressionsreihenfolge wird mit LRCP vorgegeben. Der Empfänger kann durch Anfrage und Interpretation — und unter Verwendung der vorgegebenen Parameter — die Struktur der originalen Datei nachbilden. Die Übertragungseffizienz übertrifft dabei die Ansätze aus [16] und [17], jedoch nicht das JPIP Verfahren. Allerdings eliminiert J2L viele Probleme, die bei der Verwendung von JPIP entstehen können, wie z. B. die Implementierung eines dedizierten JPIP-Servers.

5.2.1 Erweiterung des Dateisystems um parametrisierbare Dateiaufrufe

Die Motivation zu parametrisierbaren Dateiaufrufen ergibt sich aus der Beobachtung, dass Hard- und Softwarelösungen verfügbar sind, die auf der einen Seite bspw. eine Videosequenz in einer festgesetzten maximalen Auflösung (HD) in Echtzeit zu decodieren. Auf der anderen Seite sind diese Hardware-Bausteine aber nicht in der Lage, eine gewünschte Variante aus einem skalierbaren Videostrom auszulesen, wenn dieser eine zu hohe Auflösung besitzt. Hier wird es nötig, eine Zwischenvariante des Videos, das bspw. in 4K UHD (3840x2160 Pixel) vorliegt, zu erstellen, damit die Decoder-Einheit den Datenstrom anschließend verarbeiten kann. Weiter kann es der Fall sein, dass vorhandene Decoder-Lösungen nicht alle verfügbaren Skalierungsstufen unterstützen, so dass bestimmte Optionen nicht genutzt werden können. Bei der Verwendung von Hardware ist es nicht möglich, fehlende Funktionen nachzuliefern. Auch wenn dies bei Software machbar ist, müssen Änderungen üblicherweise in mehreren

Programmen nachgezogen werden. Diese Änderungen können sehr arbeitsaufwendig sein und sind darüber hinaus nur möglich, wenn Zugriff auf den Quellcode besteht.

Aus diesem Grund wurde im Rahmen dieser Arbeit ein Ansatz verfolgt, der es ermöglicht, die Skalierung an einer zentralen Stelle vorzunehmen, um dadurch die Generierung von Proxydateien weiter zu reduzieren. Hierzu erlaubt das entwickelte System, übliche Dateiaufrufe mit zusätzlichen Parametern zu versehen, die eine Skalierung der Datei bewirken können, wenn die angefragte Datei diese Skalierung erlaubt. Dabei wird die Anwendung der Skalierbarkeit in das Dateisystem verschoben, wodurch ein neues Konzept der Dateianfragen ermöglich wird. Abstrakte Anfragen an das Dateisystem können wie folgt aussehen:

- Liste alle Dateien im Ordner <*Ordnername*> mit 25% ihrer Auflösung auf.
- Öffne Bild <Pfadangabe/Bildname> mit einer maximalen horizontalen Auflösung von 512 Pixeln.
- Öffne alle Bilder im Ordner <*Ordnername*> und skaliere diese so, dass eine maximalen Datenrate von 150 MBit/s bei 24 Bildern pro Sekunde nicht überschritten wird.

Eine Betrachtung gängiger Dateisysteme zeigt, dass diese in der Regel hierarchisch in einer Baumstruktur aufgebaut sind (vgl. 2.2.3). Dies stellt sich bspw. bei Windows so dar, dass zuerst das Wurzelverzeichnis eines Datenträgers — gefolgt von einem Doppelpunkt — angegeben wird. Anschließend werden die Verzeichnisse entsprechend der Hierarchie von oben nach unten aufgelistet. Abzweigungen werden durch einen *Slash* bzw. *Backslash* (Windows) angezeigt. Der letzte Eintrag in einer Pfadangabe ist entweder der Name eines Verzeichnisses, einer Datei oder einem Selektionsmuster für die Auswahl mehrerer Elemente. Selektionsmuster beschreiben dabei eine Teilmenge der Dateien, die auf einer angegebenen Hierarchieebene ausgewählt werden sollen.

Üblicherweise folgt dem Dateinamen die Dateiendung — beide Elemente werden durch einen Punkt getrennt. Die Dateiendung erlaubt es, den Dateityp zu erkennen, ohne den Inhalt interpretieren zu müssen. Damit eine Pfadangabe korrekt interpretiert werden kann, werden diese mit Sonderzeichen versehen. Tabelle 10 zeigt hierzu reservierte Sonderzeichen für das Windows-Betriebssystem. Entsprechende Werte dürfen nicht Teil der Ordner- oder Dateinamen sein.

Zur Übergabe zusätzlicher Parameter während des Dateiaufrufes sind verschiedene Verfahren denkbar. So könnte bspw. eine Datei mit vordefiniertem Namen in dem selben Ordner wie die Zieldaten abgelegt werden. Diese Datei könnte dann dynamisch angepasste Parametersätze beinhalten, die nach Anfrage einer Datei vom Dateisystem interpretiert werden. Ist die Parameterdatei nicht vorhanden oder leer, werden entsprechend keine weiteren Operationen an den vorhandenen Dateien vorgenommen.

Tabelle 10: Reservierte Zeichen für Pfadangaben in Windows Betriebssystemen

Zeichen	Name	Funktion
<	Kleiner als	Programmeingabe
>	Größer als	Umleitung der Ausgabe
:	Doppelpunkt	Laufwerksangabe
„	Anführungszeichen	Zusammenfassung mit Leerzeichen getrennter Angabe
/	Schrägstrich vorwärts	URL
\	Schrägstrich rückwärts	Verzeichnistrennzeichen
|	Senkrechter Strich	Pipe
?	Fragezeichen	Platzhalter für beliebiges Zeichen in einem Selektionsmuster
*	Stern	Platzhalter für kein, ein oder mehrere beliebige Zeichen in einem Selektionsmuster

Innerhalb dieser Arbeit wurde jedoch ein anderes Konzept verfolgt, das eine Parameterübergabe nach dem Dateinamen ermöglicht, wie dies auch beim Aufruf einer HTTP-Seite über eine URL möglich ist. Um Konflikte mit reservierten Zeichen zu vermeiden, wurde dabei nicht das von einer URL bekannte Parametertrennzeichen ‚?', sondern ein Semikolon verwendet. Dabei trennt das Semikolon sowohl Parameter und Dateinamen, wie auch Parameter untereinander und wird somit selbst zu einem reservierten Sonderzeichen, das bei der Verwendung des angepassten Dateisystemprototypen nicht mehr zur Ordner- bzw. Dateibenennung verwendet werden darf. Als Übergabewerte sind sowohl Flags wie auch Parameter vorgesehen (vgl. Tabelle 11):

1. Flag: der Parameterwert erhält implizit den booleschen Wert *true*, wenn er im Parametersatz aufgelistet ist. Andernfalls interpretiert das System den Wert als *false.*

2. Parameter: Ein Parameter setzt einen anschließenden Wert voraus. Parametername und Parameterwert sind durch ein Ist-Gleich-Zeichen getrennt.

Neben der Parametrisierung von Dateiaufrufen ist es ebenfalls möglich, ganze Verzeichnisse mit Zusatzparametern zur Anfrage einer Subvariante zu versehen. Hierdurch werden Parameter durch einmalige Angabe für alle enthaltenen, skalierbaren Dateien angewendet. Als Einschränkung wurde dabei festgelegt, dass eine Parametrisierung auf komplette Ordner nur in der niedrigsten Hierarchiestufe stattfinden darf, da hierdurch (a) der Parsing-Schritt zum Auslesen der Parameter vereinfacht wird und (b) maximal zwei Parameterangaben pro Datei berücksichtigt werden müssten. Die erste Parametereinschränkung erfolgt in diesem Fall durch Parameterangabe auf den enthaltenen Ordner. Eine zweite, optionale Parameterangabe kann zusätzlich noch auf der Datei selbst erfolgen.

Tabelle 11: Implementierte Skalierungsoptionen für den parametrisierbaren Dateiaufruf

Parametername	Wertebereich	Beschreibung
qualitypercentage	1...100	Prozentuale Angabe der Bildqualität. Ausgangswert ist die maximale Qualität der angefragten Datei
resolutionpercentage	1...100	Prozentuale Angabe der Bildgröße. Ausgangswert ist die maximale Auflösung der angefragten Datei
componentpercentage	1...100	Prozentuale Angabe der Bildkomponenten. Ausgangswert ist die maximale Anzahl an Bildkomponenten in der angefragten Datei
maxbitrate	>0	Angabe der maximalen Bitrate in MBit/s
frameratenumerator	>0	Zähler der Bildrate
frameratedenominator	>0	Nenner der Bildrate
maxwidth	>0	Angabe der maximalen Bildbreite
maxheight	>0	Maximale Anzahl an Pixeln in vertikaler Auflösung
maxsize	>0	Maximale Dateigröße pro Datei in Byte
levels	>0	Anzahl der Auflösungsstufen in einer Mediendatei
layers	>0	Anzahl der Qualitätsschichten in einer Mediendatei
components	>0	Anzahl der Komponenten in einer Mediendatei

Tabelle 12 zeigt jeweils ein mögliches Beispiel für die geschilderten Parametrisierungsmöglichkeiten. Dabei gibt der Parameter *maxwidth* einen absoluten Wert an. Die Bildbreite darf den Wert 512 nicht überschreiten, kann aber durchaus geringer sein. Der Parameter *qualitypercentage* steht für einen prozentualen Wert. Im gegebenen Beispiel wird die Qualität der gespeicherten Bilder demnach um mindestens 50% von den Varianten reduziert, auf die der angemeldete Benutzer zugreifen kann. Abschließend zeigt Tabelle 12 eine Kombination der Parameterübergabe für einen Ordner und eine konkrete Datei. Dabei wird die Qualität aller skalierbaren Bilder im Ordner *X:\ScalableImages* um mindestens 50% reduziert. Auch hier ist ausschlaggebend, welche maximale Qualität der Benutzer anhand der vergebenen Zugriffsrechte erhält (vgl. 4.1). Anschließend wird erneut über den Parameter *maxwidth* dafür gesorgt, dass das Bild *image001.j2*c den Wert für die horizontale Auflösung von 512 Pixeln nicht überschreitet.

Dabei erfolgt die Auswertung und Anwendung der übergebenen Parameter in hierarchischer Reihenfolge, gemäß der Struktur heutiger Dateisysteme, wenn eine kombinierte Übergabe auf Datei- und Ordnerebene gewünscht ist. Listing 5 zeigt hierzu eine Beispielanfrage. Die Bilder im Ordner *X:\ScalableImages* bieten eine Auflösung von 4K und der Benutzer besitzt

Tabelle 12: Beispiele zur Parameterübergabe auf Dateien, Ordner sowie der kombinierten Anwendung auf Dateien und Ordner

Anwendung auf	Beispiel
Dateiebene	image001.j2c;maxwidth=512
Verzeichnisebene	X:\ScalableImages;qualitypercentage=50
Datei-& Verzeichnisebene	X:\ScalableImages;qualitypercentage =50\image001.j2c;maxwidth=512

entsprechende Zugriffsrechte auf die volle Auflösung der Bilder. Eine Interpretation der Anfrage aus Listing 5 ist in mindestens zwei Varianten denkbar:

1. Das System erkennt, dass der Parameter *resolutionpercentage* sowohl im Verzeichnis- wie auch im Dateinamen vergeben ist. Weiter erkennt das System, dass in beiden Parametern die gleiche Reduktion gewünscht ist und geht im Folgenden davon aus, dass der Benutzer die ursprüngliche Auflösung insgesamt um 50% reduzieren möchte. Demnach wird die 4K-Variante um 50% reduziert und der aufrufende Prozess erhält die entsprechende 2K-Variante.

2. Das System wendet den übergebenen Parameter *resolutionpercentage* zweimal an. Eine erste Anwendung erfolgt nach Auswertung der Parameter für den beinhalteten Ordner. Als Resultat steht intern eine 2K-Variante zur Verfügung, die nachfolgend um die Parameter zum Dateinamen reduziert wird. Entsprechend wird die 2K-Zwischenvariante ein weiteres Mal in ihrer Auflösung um 50% reduziert, wodurch eine 1K-Variante der skalierbaren Bilddatei entsteht.

Die aktuelle Implementierung folgt dabei dem zweiten Ansatz. Entsprechend kaskadieren sich doppelt vergebene Parameter im Verzeichnis- und Dateinamen. Dabei ist zu beachten, dass eine zweite Auswertung eines bereits auf Verzeichnisebene verwendeten Parameters keine Erweiterung der bereits eingeschränkten Variante mehr erlaubt. Listing 6 soll dies verdeutlichen: Der Parameter *maxwidth* gibt hierbei einen Absolutwert für die maximale horizontale Auflösung an. Bei der Parametrierung des Verzeichnisses wird dieser Wert bereits auf 512 Pixel beschränkt. Entsprechend kann eine Parametrierung der Datei den Wert von 512 lediglich weiter reduzieren, bereits angewendete Einschränkungen aber nicht erweitern, auch wenn der aktuelle Benutzer entsprechende Zugriffsrechte besitzt, um das Bild *image001.j2c*

Zum Parsen der definierten Parameter wurde im Rahmen dieser Arbeit ein endlicher Automat entwickelt und anschließend implementiert. Die detaillierte Beschreibung zur Entwicklung des entstehenden Automaten ist in Anhang D gezeigt. Die Definition der Zustände und in einer horizontalen Auflösung von 1024 Pixel anzufragen.

```
X:\ScalableImages;maxres=50\image001.j2c;maxres=50
```

Listing 5: Beispiel einer parametrisierten Dateianfrage auf Datei- und Verzeichnisebene

```
X:\ScalableImages;maxwidth=512\image001.j2c;maxwidth=1024
```

Listing 6: Beispiel einer parametrisierten Dateianfrage mit absoluten Werten auf Datei- und Verzeichnisebene

Tabelle 13: Formale Grammatik zum Parsen der Parameter für skalierbare Medien

<ROOT>	→	\<UV_NAME>
<UV_NAME>	→	C<UV_NAME_REST>
<UV_NAME_REST>	→	C<UV_NAME_REST>|\<UV_NAME>|;<UV_PARAM>|<ε>
<UV_PARAM>	→	P<UV_PARAM_REST>
<UV_PARAM_REST>	→	P<UV_PARAM_REST>|;<UV_PARAM>|=<UV_WERT>| \<DNAME>|<ε>
<UV_WERT>	→	P<UV_WERT_REST>
<UV_WERT_REST>	→	P<UV_WERT_REST>|;<UV_PARAM>|\<DNAME>|<ε>
<D_NAME>	→	C<D_NAME_REST>
<D_NAME_REST>	→	C<D_NAME_REST>|;<D_PARAM>|<ε>
<D_PARAM>	→	P<D_PARAM_REST>
<D_PARAM_REST>	→	P<D_PARAM_REST>|;<D_PARAM>|=<D_WERT>|<ε>
<D_WERT>	→	P<D_WERT_REST>
<D_WERT_REST>	→	P<D_WERT_REST>|;<DPARAM>|<ε>

Übergänge erlaubt die Definition einer formalen Grammatik zum Parsen der Parameter. Die Grammatik lässt sich dabei direkt aus Abbildung 60 und Abbildung 61 ableiten und ist in Tabelle 13 dargestellt.

5.2.2 Implementierung

Mit Hilfe der formalen Grammatik lässt sich eine Entscheidungstabelle erstellen, die für den implementierten Parser in Tabelle 14 dargestellt ist. In der horizontalen Achse sind dabei die möglichen Kanten angegeben, die vertikale Achse beinhaltet die möglichen Zustände des Automaten. Leere Zellen in Tabelle 14 entsprechen dem Fehlerzustand. Die Erweiterungen zur parametrisierbaren Dateianfrage wurden auf dem Prototypen eines virtuellen Dateisystems für skalierbare Mediendaten (vgl. 4.1.1) entwickelt. Dabei sollen die in Tabelle 11 gezeigten Skalierungsstufen über zusätzliche Parameterangaben einstellbar sein. Die erstellte Entscheidungstabelle kann dabei einfach in Form von *Switch-Case*-Anweisungen in C/C++ Quellcode übertragen werden. Beispielhaft zeigt Listing 7, dass jeder *case*-Abschnitt einen Zustand des entwickelten Automaten darstellt. Innerhalb der *if/else*-Konstrukte werden dann die unterschiedlichen Zustandsübergänge abgebildet.

Tabelle 14: Entscheidungstabelle für den endlichen Automaten zum Parsen der Parameter bei einem parametrierten Dateiaufruf

Zeichen / Zustand	\	C	;	ε	P	=
ROOT	UV_NAME					
UV_NAME		UV_NAME_REST				
UV_NAME_REST	UV_NAME	UV_NAME_REST	UV_PARAM	ENDE		
UV_PARAM					UV_PARAM_REST	
UV_PARAM_REST	DNAME		UV_PARAM	ENDE	UV_PARAM_REST	UV_WERT
UV_WERT					UV_WERT_REST	
UV_WERT_REST	DNAME		UV_PARAM	ENDE	UV_WERT_REST	
D_NAME		D_NAME_REST				
D_NAME_REST		D_NAME_REST	D_PARAM	ENDE		
D_PARAM					D_PARAM_REST	
D_PARAM_REST			D_PARAM	ENDE	D_PARAM_REST	D_WERT
D_WERT					D_WERT_REST	
D_WERT_REST			D_PARAM	ENDE	D_WERT_REST	

```
01:   state nextState=ROOT; //set start state to root
02:   do {
03:       switch ( nextState ) {
04:           case ROOT:
05:               if (filename == '\\')
06:                   nextState = UV_NAME;
07:               else
08:                   nextState = ERROR;
09:              break;
                  ...
10:           case UV_WERT_REST:
11:               if (filename == '\\') nextState = D_NAME;
12:               else if(filename == ';') nextState = UV_PARAM;
13:               else if(filename == 'P') nextState = UV_WERT_REST;
14:               else if(filename == '\0') nextState = ENDE;
15:               else nextState = ERROR;
16:               break;
                  ...
17:           case ERROR:
18:               cerr << „Error during parsing process.“ << endl;
19:               return -1;
20:           case ENDE:
21:               cout << „Parsing finished successful.“ << endl;
22:               return 0;
23:       }
24:   } while (filename++)
```

Listing 7: Ausschnitt aus dem Quellcode zum Parsen zusätzlicher Parameter bei einem Dateiaufruf

Ergebnisse:

Die aktuelle Implementierung erlaubt die Übergabe von weiteren Parametern bspw. über die Kommandozeile. In Abbildung 43 ist die Vorgehensweise dazu gezeigt. Der Prototyp des virtuellen Dateisystems ist hierzu unter dem Laufwerksbuchstaben Y: erreichbar. Zunächst navigiert der Benutzer in der Kommandozeile in den virtuellen Ordner mit seinem Benutzernamen (hier: *sbg*). Anschließend folgt ein weiterer *cd*-Aufruf für den Ordner *JPEG2000_4K*, zunächst noch ohne Angabe zusätzlicher Parameter. Als nächstes wird ein übliches *directory-listing* mit dem *dir*-Befehl ausgeführt. Es folgt eine Auflistung von zehn JPEG 2000-Dateien mit einer durchschnittlichen Dateigröße von etwa 8 MB. Anschließend wird der aktuelle Ordner wieder verlassen und in das übergeordnete Verzeichnis gewechselt (*cd ..*). Danach erfolgt der Aufruf *cd JPEG2000_4K;resolutionpercentage=50* — wodurch ein parametrisierter Aufruf ausgelöst wird. In diesem Beispiel sorgt der Aufrufer dafür, dass alle im Ordner enthaltenen, skalierbaren Bilder mit 50% ihrer eigentlichen Auflösung dargestellt werden. Ein weiteres *directory-listing* zeigt, dass die Dateinamen der enthaltenen JPEG 2000-Bilder identisch sind, die Dateigröße allerdings reduziert ist (~3,6MB), was auf die gewünschte Reduktion der Bildgröße zurückzuführen ist.

Neben einem Aufruf in der Kommandozeile lässt sich das Konzept des parametrisierbaren Dateiaufrufes auch direkt in entsprechender Bildbetrachtungssoftware nutzen. Abbildung 44 zeigt hierzu das Verhalten von *IrfanView* [100] unter dem Betriebssystem *Windows*. Als Einstiegspunkt dient erneut der Laufwerksbuchstabe *Y*. Anschließend wurde über die grafische Benutzeroberfläche in den Ordner *JPEG2000_4K* navigiert. Dabei wurden keine Parameter beim Aufruf der Ordner verwendet. In Abbildung 44 wird das erste Bild der Sequenz angeklickt, was dazu führt, dass IrfanView das Bild für eine Vorschau decodiert und die verfügbaren Metadaten anzeigt (Abbildung 44 unten links). Es ist zu erkennen, dass das Bild eine 4K-Dimension (4096x2160) aufweist. Im Feld *Dateiname* wird zusätzlich der Parameter *maxwidth* mit einem Wert von 512 hinzugefügt, wodurch der Aufruf der Datei derart parametriert wird, dass die maximale horizontale Auflösung auf diesen Wert beschränkt wird. Anschließend wird die Datei geöffnet, das Ergebnis ist in Abbildung 45 zu sehen. Die Software zeigt dabei den Pfad zur geöffneten Datei an. Der Titel des Fensters beinhaltet neben dem Dateinamen noch die zusätzlichen Parameter. Die Metadaten in der unteren Zeile des Anzeigefensters machen deutlich, dass die Parametrierung erfolgreich war, da das aktuell gezeigte Bild eine Auflösung von 512x270 Pixeln aufweist.

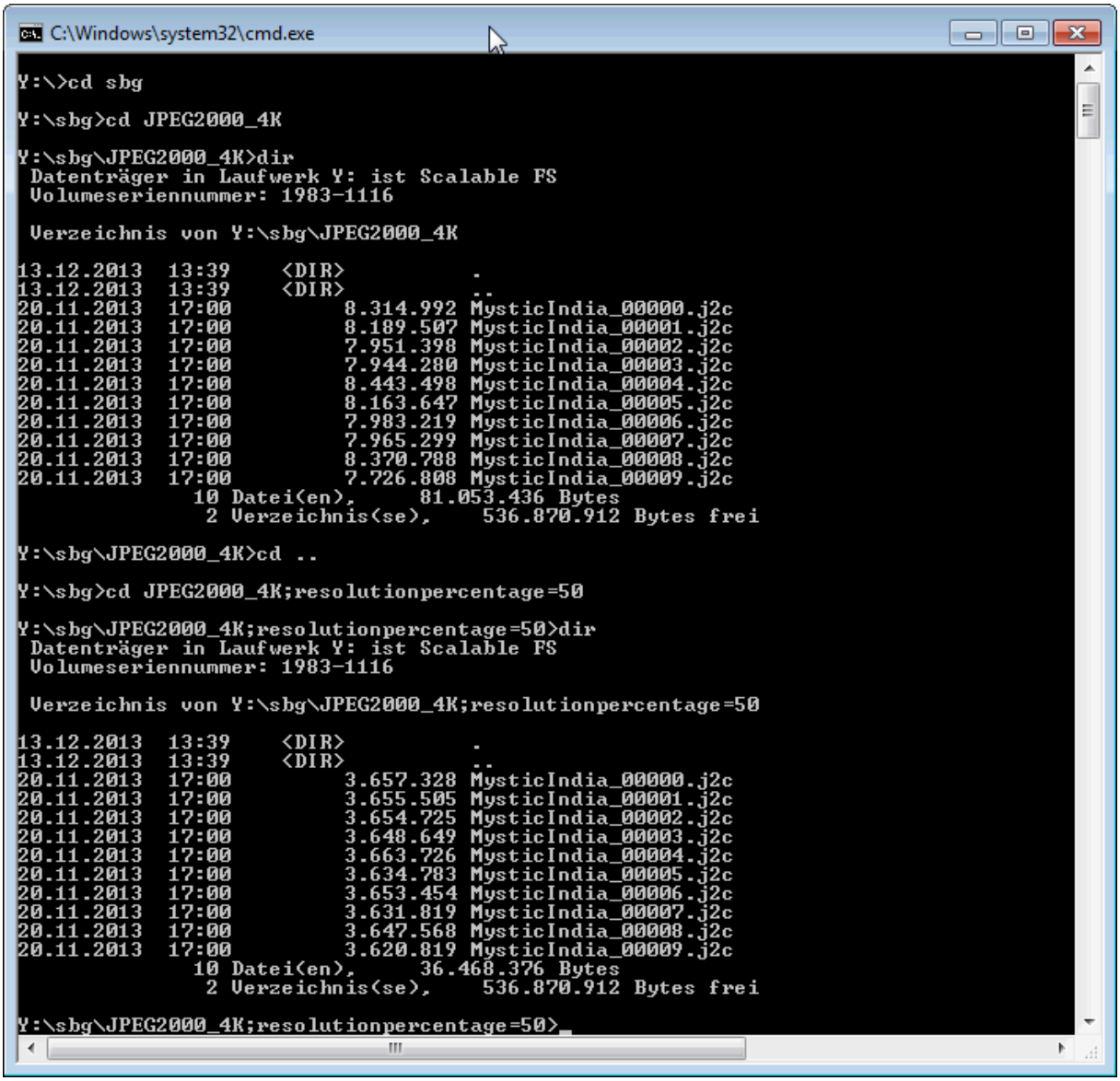

Abbildung 43: Anzeige aller Dateien in einem Ordner mit Angabe zusätzlicher Parameter

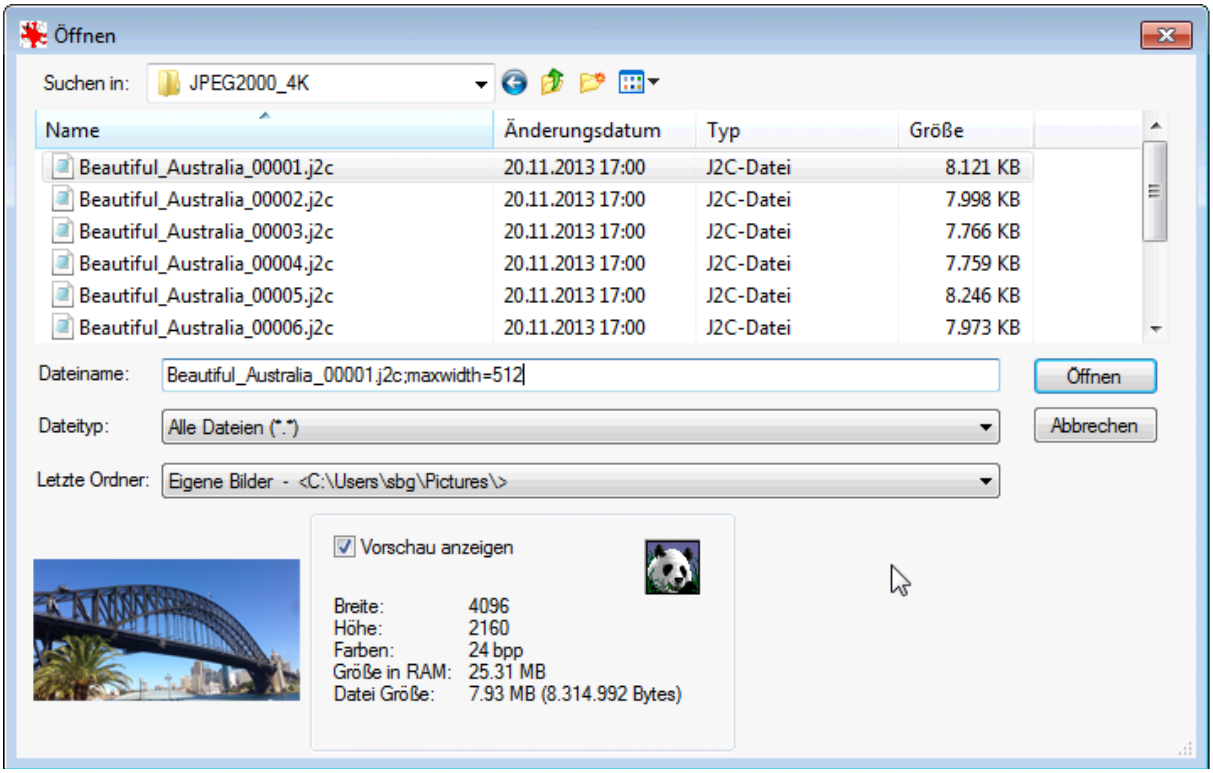

Abbildung 44: Anwendung parametrisierbarer Dateiaufrufe in der Dekoding-Software IrfanView

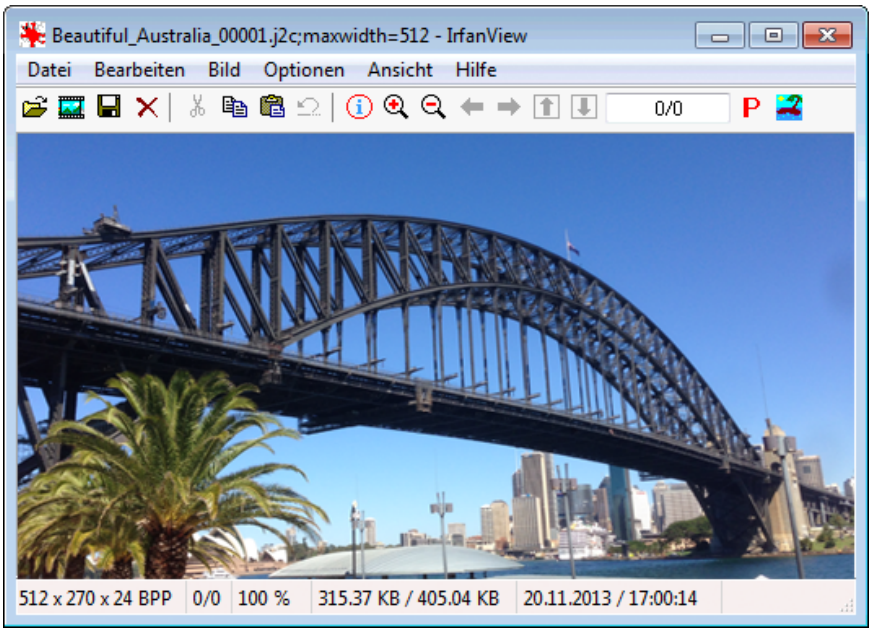

Abbildung 45: Ergebnis eines parametrierten Dateiaufrufs in der Software IrfanView

5.3 Cache-Verfahren

[18]Cache-Verfahren umfassen im Wesentlichen eine Einlagerungs- und eine Ersetzungsstrategie. Bei der Einlagerungsstrategie wird bestimmt, welche Daten im Cache gespeichert werden. Die Ersetzungsstrategie hingegen entscheidet, welche Daten aus dem Cache entfernt werden müssen, wenn nicht genug Speicher vorhanden ist, um neue Daten aufzunehmen. Heutige Cache-Verfahren in Consumer-Betriebssystemen besitzen keine Kenntnis über die Skalierbarkeit moderner Medienkompressionsverfahren und nutzen potenzielle Verbesserungsmöglichkeiten demnach nicht aus. Deshalb werden in diesem Abschnitt sowohl Einlagerungs- wie auch Ersetzungsstrategien für skalierbare Medien vorgestellt. Nach aktuellem Kenntnisstand existieren keine Vorarbeiten, die dieses Thema konkret bearbeiten. Jedoch finden sich wissenschaftlichen Arbeiten, die sich mit Prefetching- und Cache-Verfahren skalierbarer Medien im Bereich von Computernetzen beschäftigen. Prefetching wird dabei verwendet, um Caches mit bestimmten Daten vorzufüllen um damit z. B. Übertragungsengpässe mit diesen präventiv gesendeten Daten zu überbrücken.

5.3.1 Relevante Vorarbeiten

Relevante Arbeiten im Bereich von Einlagerungs- sowie Prefetching-Strategien sind [30,86,87,88,134,135,136,137], die im Folgenden beschrieben werden. [30] vergleicht adaptive Streaming-Verfahren mit skalierbaren Videos im Bereich Video-on-Demand. Es zeigt sich, dass skalierbare Medien weniger Datentransfer verursachen, wenn die Anfragen der Nutzer sehr inhomogen ausfallen. Hier empfehlen die Autoren ein popularitätsbasiertes Vorfüllen des Proxy-Caches unter Verwendung eines skalierbaren Codecs zur weiteren Optimierung des Gesamtsystems. Die Arbeiten [86] bzw. [87] basieren auf der Beobachtung, dass häufige Qualitätsänderungen während des Abspielens skalierbarer Videos von Betrachtern als qualitativ minderwertig empfunden werden. Deshalb stellen die Autoren ein qualitätsadaptives Verfahren vor, das zunächst noch ohne Proxy-Server arbeitet. Durch das Netzwerk bedingte Durchsatzschwankungen werden hier durch einen Puffer beim Empfänger kompensiert, der präventiv so gefüllt wird, dass die Qualitätsschichten der skalierbaren Videos [54,66,67,124] über einen bestimmten Bereich konstant sind. Das präventive Füllen erfolgt immer dann, wenn mehr Bandbreite zur Verfügung steht, als für das aktuelle Abspielen benötigt wird. Hierdurch kann ein Video bei sinkender Bandbreite für einen gewissen Zeitraum mit konstanter Qualität abgespielt und Qualitätswechsel somit minimal gehalten werden. In [89] wird die vorgestellte Architektur noch um einen Proxy-Server erweitert. Dabei erhält auch der Proxy einen sog. *Prefetching-Algorithmus*, der Daten bei Bedarf vorschauend anfragt. Hierbei ist es allerdings ein weiteres Ziel, die Qualität von

[18] Teilaspekte dieses Abschnitts wurden in [105] sowie [109] veröffentlicht.

Videos durch einen Cache im Proxy sukzessive zu verbessern, wenn diese mehrfach abgespielt werden. Dabei werden bereits empfangene Qualitätsschichten vom Proxy geliefert und — wenn nicht alle angefragten Daten im Proxy vorhanden sind — zusätzliche Qualitätsschichten vom Server nachgeladen. Der Server liefert angefragte Segmente priorisiert zurück, indem zunächst fehlende Segmente der untersten Qualitätsschicht versendet werden. Hierdurch weisen häufig angefragte Videos eine höhere Qualität im Proxy-Cache auf als weniger populäre Videos.

Zink et al. untersuchen die visuelle Qualität beim Cachen von skalierbaren Videos, die beim Betrachter ankommt. In [134] wurde durch visuelle Testverfahren ermittelt, dass die wahrgenommene Qualität skalierbarer Videos besonders zunimmt, wenn (i) die Datenmenge der untersten Qualitätsschicht erhöht wird und (ii) fehlende Segmente in den unteren Qualitätsschichten minimal sind. Diese subjektiven Ergebnisse ließen sich allerdings nicht über den PSNR-Wert nachstellen, weshalb die Autoren in [135] bzw. [136] den sog. *Spectrum*-Wert vorschlagen, der sowohl die Frequenz wie auch die Amplitude der Qualitätsvarianten während des Abspielens von skalierbaren Videos berücksichtigt. Wird ein bereits teilweise zwischengespeichertes Video erneut angefragt, werden fehlende Segmente folglich immer so nachgefordert, dass der *Spectrum*-Wert minimal[19] wird. Die Wahl fällt auch hier auf ein heuristisches Modell, da in [137] gezeigt wurde, dass ein optimales Verfahren zu rechenintensiv ist. Die verwendete Heuristik basiert auf einer Priorisierung fehlender Pakete mit dem Ziel, Lücken einer Auflösungsstufe im Proxy-Cache zu schließen. Hierzu werden die vorhandenen Lücken zunächst nach Länge und anschließend nach Höhe der Qualitätsschicht sortiert. Die Simulationen zeigen, dass die Qualität — gemessen am *Spectrum*-Wert — erheblich steigt.

In [52] schlagen die Autoren eine Cache-Ersetzungsstrategie für nicht-skalierbare Einzelbilder vor. Auch hier wird eine Netzwerktopologie mit Proxy-Server betrachtet. Im Falle eines volllaufenden Caches im Proxy werden die Bilder nicht direkt verworfen, sondern nach Möglichkeit mit einer geringeren Datenrate neu encodiert. Hierdurch wird Speicher im Cache freigegeben, wodurch neue Daten hinzugefügt werden können. Nachteilig ist hier der Aufwand für die Neucodierung, allerdings verweisen die Autoren auf die Möglichkeit, die Verfahren mit skalierbaren Medien zu nutzen.

Relevante Vorarbeiten im Bereich der Ersetzungsstrategien für skalierbare Medien im Web-Umfeld sind [9,10,55,61,62,63,83,88,111] sowie [130,133,134,135,136,137], die im Folgenden kurz erläutert werden. Ausgehend davon, dass beim LRU-Verfahren Caches mit Dokumenten überflutet werden können, die nur einmal angesehen werden, stellen die Autoren in [111] den *Scalable Time To Live Algorithmus (STTL)* vor. Dieser Algorithmus weist jeder Qualitätsschicht ein Zugriffswert zu, der sowohl Zugriffsfrequenz wie auch Zeitpunkt des

[19] Die wahrgenommene Qualität erhöht sich bei abnehmendem *Spectrum*-Wert.

letzten Zugriffs berücksichtigt. Die Verwendung von STTL reduziert die Netzwerklast und zeigt eine schnellere Antwortzeit bei gleicher Cache-Größe im Vergleich zum LRU-Verfahren. Chiu und Yeung [10] zeigen ein größenadaptives Cache-Verfahren, das nur bestimmte Teile eines Video-Streams speichert, um dadurch Bandbreite zwischen Server und Proxy-Cache zu sparen. Beim verwendeten, skalierbaren Videoformat [9] werden nur große Segmente gecacht, da kleine Datenmengen bei einer erneuten Anfrage vom Videoserver übertragen werden können. In [89] wird neben der bereits genannten Prefetching-Strategie eine Ersetzungsstrategie vorgestellt. Hier wird eine grobkörnige Ersetzung durch Entfernung einer kompletten Qualitätsschicht der unpopulärsten Videos erreicht. Die Simulationen zeigen, dass die Kombination aus Prefetching- und Ersetzungs-Algorithmik den Zustand des Caches näherungsweise in einen idealen Zustand versetzen. Eine Implementierung dieser Verfahren in einer Client-/Server-Architektur [88] speichert jede Qualitätsschicht eines Videos in einer separaten Datei. Hierdurch wird die Nutzung bestehender Cache-Verfahren vorhandener HTTP-Server gewährleistet. Beim *Quality-Based-Caching* [83] liegt der Fokus auf Ersetzungsstrategien bei volllaufendem Cache. Neben der vertikalen Ersetzung, bei der auf Grund von gemessenen Popularitäten immer die Qualitätsschicht mit der geringsten Popularität entfernt wird, wird die *horizontale Ersetzung* vorgestellt. Dabei werden alle aktuell gecachten Videos zusammen betrachtet und bei Bedarf die höchste Qualitätsschicht aller Videos verworfen, wenn Speicher freigegeben werden muss. Die Simulationsergebnisse zeigen, dass die Trefferrate sehr gut ist, wenn eine horizontale Verwerfung durchgeführt wird. Ein kombiniertes Verfahren aus horizontaler und vertikaler Ersetzung steigert sowohl die Hit-Rate im Vergleich zur vertikalen Ersetzung wie auch die *Quality-Weighted-Hit-Rate* im Vergleich zu allen vorgestellten Verfahren, wenn die Popularität der Videos nicht zu stark verteilt ist. Unter Berücksichtigung der begrenzten Faktoren Bandbreite und Cache-Größe wird in [51] untersucht, welche Teile eines skalierbaren Videos im Idealfall in einem Cache zwischengespeichert werden müssen. Das verwendete, heuristische Modell liefert hierzu eine Abschätzung, welche Qualitätsschichten der Videos (hier: [8,54,67,124]) in Abhängigkeit ihrer Popularitäten vorgehalten werden sollten. Dabei werden die zwischengespeicherten Daten im Proxy-Server zyklisch durch den Server aktualisiert. Um die populären Qualitäts-schichten zu finden, wird für jede Stufe ein *Cache-Utility (CU)* berechnet, der wahlweise (i) die Popularität, (ii) die Popularität multipliziert mit der Qualitätssteigerung oder (iii) die Popularität multipliziert mit der Qualitätssteigerung, gewichtet mit der verfügbaren Bandbreite, berücksichtigt. Beim nutzeradaptiven und bandbreitenadaptiven Verfahren für einen Proxy-Server werden in [63] vorhandene Qualitätsschichten in so genannte Segmente unterteilt, denen anschließend ein Popularitätsindex zugewiesen wird. Zur Adaption typischer Nutzerverhalten wird hier folgendes Regelwerk angewendet: (i) Basis-Qualitätsschichten bekommen eine höhere Priorität als Detailqualitätsschichten, (ii) Segmente, die am Anfang des Videos sind, erhalten eine höhere Priorität als nachfolgende Segmente und (iii) ein Segment bekommt eine hohe Priorität, wenn die Bandbreite vieler Klienten ausreicht um eine Qualität abzufragen, die das entsprechende Segment beinhaltet. Bei extrem geringen Cache-

Größen im Proxy kann das Verfahren eine Verbesserung erreichen, da der Cache zu klein sein kann, um eine komplette Qualitätsschicht zu speichern, aber dennoch groß genug ist, um Segmente zu speichern.

Neben der bereits vorgestellten Einlagerungsstrategie zeigen Zink et al. in [135], wie eine sog. *Glättung* (engl. *Polishing*) innerhalb einer Cache-Ersetzungsstrategie zur weiteren Steigerung der empfundenen Qualität beitragen kann. Idee ist, aufbauend auf einer Ersetzungsstragie nach Priorität in [133], eine Kombination mit dem *Spectrum*-Parameter herzustellen. Hierzu wurden Simulationen durchgeführt, die zeigen, dass mit dieser Technik sehr feinkörnige Ersetzungen vorgenommen werden können, die zudem vom Betrachter als sehr hochqualitativ empfunden werden. Unter Verwendung der Fine-Grained Scalable Postratenkontrolle (FGS – vgl. [55]) zur Anpassung bereits encodierter MPEG-4 Ströme an die aktuell verfügbare Bandbreite, wird in [61] bzw. [62] ein netzwerkadaptives Video-Caching-Framework vorgestellt. Server und Proxy sind hier in der Lage, bereits encodierte Videos nachträglich so zu bearbeiten, dass die Datenrate adaptiv an die verfügbare Bandbreite angepasst werden kann. Weiter stellen die Autoren eine Cache-Einlagerungsstrategie vor, die berücksichtigt, welche Daten noch in Echtzeit vom Server nachgeladen werden können. Somit werden Qualitätsschichten im Proxy-Cache ersetzt, um Speicherplatz für neue Daten freizugeben, wenn das Gesamtsystem in der Lage ist, diese Daten nachzuliefern. Eine weitere Art der Skalierung und deren Nutzung zeigt die Arbeit [130], in der vorhandene Videos in *Prefix* und *Suffix* unterteilt werden. Dadurch wird berücksichtigt, dass häufig nur der erste Teil eines Videos betrachtet wird. Ziel ist es demnach, möglichst viele Prefixe populärer Videos im Proxy-Cache zu speichern. Während des Streamings der Prefixe können Suffixe vom Server nachgeladen werden.

Die Anwendung skalierbarer Medien in Peer-to-Peer Netzwerken (P2P) wird in [19] untersucht. Die Autoren schlagen vor, Basis- und Erweiterungsstufen — je nach Verfügbarkeit — von den vorhandenen Netzwerkteilnehmern (die sog. *Gruppe*) in einem Cache im Empfänger zu bündeln, bevor dieser Cache anschließend in der Lage ist, das Video neu zusammenzustellen. In [29] wird jedem Segment einer skalierbaren Datei ein Nützlichkeits-Index zugewiesen, der berücksichtigt, (i) wie teuer die Speicherung für das Segment ist und (ii) wie hoch die zukünftige Zugriffsfrequenz auf dieses Segment geschätzt wird. Die Kosten zur Speicherung berücksichtigen dabei, wie viele Kopien des Segmentes in benachbarten Knoten bereits vorliegen und wie die Bandbreite zu diesen Knoten ist. Weiter wird die Auslastung und Verfügbarkeit des Servers berücksichtigt. Je höher der Index, desto wichtiger das Segment und desto unwahrscheinlicher ist es, dass dieses Segment aus dem Cache verdrängt wird. Eine Ausnutzung der Skalierbarkeit in verteilten Streaming-Anwendungen wird in [59] untersucht. Dabei tauschen die Klienten untereinander zwischengespeicherte Daten aus, um die Anfragen an den Server zu minimieren. Die Skalierbarkeit wird dabei so genutzt, dass nicht Videos, sondern einzelne (Qualitäts-) Stufen der Videos immer dann im Proxy und Klienten gespeichert werden, wenn Daten vom Server eingehen. Die Übertragung

skalierbarer Medien (besonders SVC) über das WiMAX-Protokoll (Worldwide Interoperability for Microwave Access) und deren Herausforderungen im Bereich IPTV-Mulitcast werden in [129] betrachtet. Sanchez et al. betrachten in [94] bzw. [95] eine Optimierungsmöglichkeit des weit verbreiteten MPEG-DASH-Verfahrens [44] und vergleichen hierzu H.264 SVC mit H.264 AVC (Advanced Video Coding), um zu zeigen, dass sowohl schnellere Antwortzeiten wie auch eine geringere Latenz nach Auswahl eines Videos mit SVC erreicht werden können. Dabei wird in [94] besonders der Einfluss einer sinkenden Netzwerkverfügbarkeit während des Abspielens untersucht. Weiter nutzt die Architektur dieser Arbeit ein Cache-Verfahren in Proxy-Servern, das einen auf MGS beruhenden Caching-Algorithmus beinhaltet. Die Caching-Algorithmen sind LRU und CC [34] — ein Verfahren, das auf einem Scoring-Algorithmus für Videodaten beruht. Es werden Cache-/Hit-Ratio auf Chunk-Basis im Proxy-Cache, sowie ein Verhältnis zwischen angefragter und gelieferter Datenrate (desired/retrieved – D/R) ermittelt. Ein Chunk ist dabei ein Teil eines Videos. Zum Test werden 90-minütige Filme in Chunks von zehn Sekunden Länge aufgeteilt. Es zeigt sich, dass besonders bei Netzwerkengpässen zwischen Proxy und Klient das SVC-Format deutlich bessere Ergebnisse liefert. Die Tatsache, dass alle Betrachter eines Films das identische Basis-Layer eines Videos konsumieren, trägt hierzu bei. In [95] erweitern die Autoren ihre Überlegungen um den Aspekt der Dateiorganisation. Hierzu schlagen sie vor, SVC Dateien im H.264 AVC-Dateiformat [40] zu speichern. Dabei werden sog. *Datei-Subsets* verwendet, die einzelne Qualitätsschichten der SVC-codierten Videos speichern. Je nach Netzwerkverfügbarkeit können Klienten entsprechend mehr oder weniger Datei-Subsets nachladen. Aufbauend auf den Arbeiten von Sanchez et al. wird in [71] bzw. [72] die Adaption von SVC für mobile Anwendungen untersucht. Müller et al. zeigen, dass SVC auf Grund der Skalierbarkeit (a) eine bessere Auffüllung eines Datenpuffers ermöglicht, der in einem mobilen Datenempfänger untergebracht ist und (b) die Bandbreite zwischen Sender um Empfänger ebenfalls besser ausgenutzt werden kann.

5.3.2 Cache-Einlagerungsstrategie

Eine Cache-Strategie in Verbindung mit dem echtzeitfähigen Einlesen der skalierbaren Daten (5.1) stellt eine sinnvolle Weiterentwicklung der vorgestellten Algorithmen dar. Dabei werden die gelesenen Anteile der Bilder, auf die während einer Session mehrmals zugegriffen wird, bei jedem neuen Einlesevorgang erhöht. Dies kann bspw. der Fall sein, wenn eine kurze Szene innerhalb der Postproduktion in einer Schleife (engl. *Loop*) abgespielt und deshalb immer wieder eingelesen wird. Dabei ermittelt das Dateisystem, ob ein Bild bereits komplett aus dem eigenen Cache bedient werden kann oder ob eine weitere Leseanfrage an den Massenspeicher gestellt werden muss. Bei Zugriff auf den Datenträger gilt weiterhin die Echtzeitbedingung, so dass das Gesamtsystem eine gültige Bilddatei innerhalb einer vorgegebenen Zeit liefert. Abbildung 46 zeigt das prinzipielle Verhalten der zugeschalteten Cache-Strategie: Analog zu der Substitutionsmethode wird durch das Auslesen der

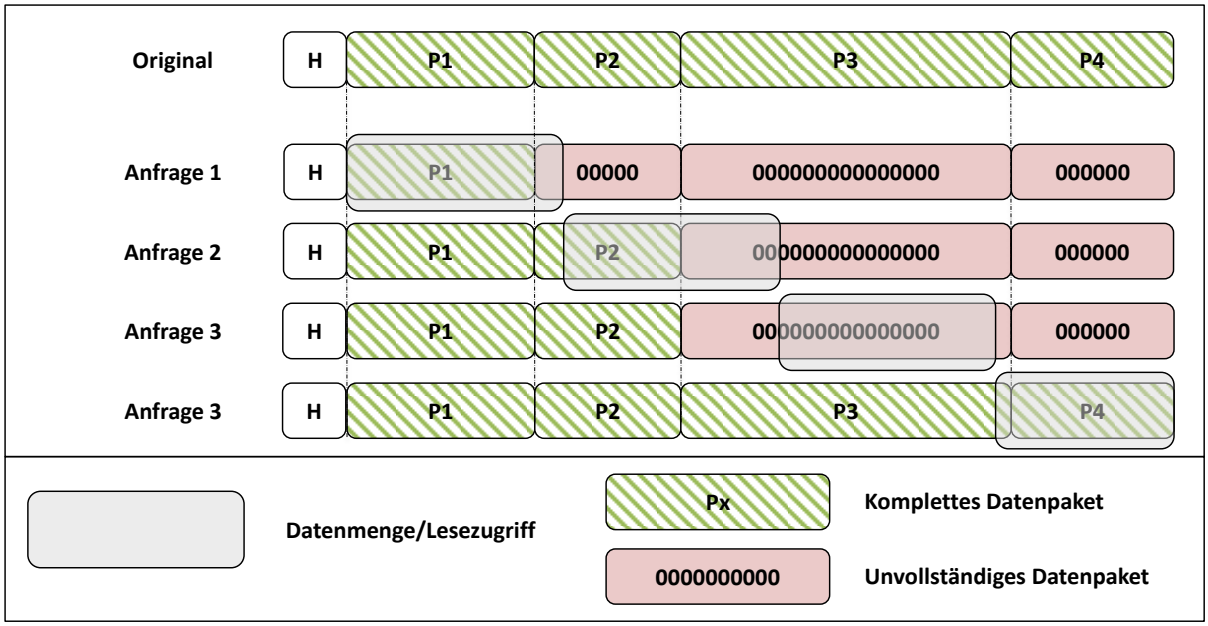

Abbildung 46: Caching-Strategie mit vier Anfragen auf eine skalierbare JPEG 2000-Datei

Header-Informationen die Struktur des komprimierten Bildes im Speicher nachgebaut (Abbildung 46, Anfrage 1), wenn noch keine Variante der Datei vorhanden ist. Das Dateisystem liest bei der ersten Anfrage noch weitere Daten, die in diesem Beispiel ausreichen, um das erste Paket (P1) komplett einzulesen. Alle gelesenen Daten — also in diesem Fall auch ein Teil von P2 — werden für weitere Aufrufe im Cache vorgehalten. Pakete, die innerhalb der vorgegebenen Zeit komplett gelesen werden konnten, werden an den aufrufenden Prozess zurückgeliefert. Pakete, die nur teilweise oder gar nicht eingelesen werden konnten, werden gemäß der Substitutionsmethode durch Nullpakete ersetzt. So liefert die erste Anfrage (Abbildung 46, Anfrage 1) ausreichend Informationen, um das erste Paket zu komplettieren. Alle gelesenen Daten werden im Cache des Dateisystems vorgehalten, ohne dabei Rücksicht auf die Paketgrenzen der skalierbaren Datei zu nehmen. Nach dem ersten Zugriff wurden viele Daten der originalen Datei substituiert — entsprechend ist die Qualität der virtuellen Datei im Vergleich zur originalen Datei deutlich geringer (vgl. Abbildung 17-a, Version 1). Bei einem weiteren Zugriff auf die gleiche Datei liest das Dateisystem an genau der Stelle weiter, an der der letzte Aufruf auf Grund der Echtzeitanforderung unterbrochen wurde (Abbildung 46, Anfrage 2). Als Resultat können P1 und P2 an den aufrufenden Prozess geliefert werden und die Qualität des Bildes steigert sich entsprechend. Der dritte Aufruf liefert nicht ausreichend Daten, um P3 komplett einzulesen. Entsprechend liefern der zweite und dritte Aufruf die gleiche Variante der virtuellen Datei, obwohl die gecachte Variante der Datei verbessert wurde. Erst der vierte Aufruf (Abbildung 46, Anfrage 4) liefert genug Daten, um P3 und P4 zu vervollständigen.

5.3.3 Evaluierung und Ergebnisse

Zur Evaluierung der Cache-Strategie wurde die bereits bei der echtzeitfähigen Einlesestrategie vorgestellte Testsequenz und das entsprechende Testprogramm verwendet. Getestet wurden hier drei verschiedene Bildwiederholraten, die zwangsläufig die Einlesezeit pro Aufruf einer Datei festlegen. Der Vergleich der lokal gespeicherten, virtuellen JPEG 2000-Dateien und der originalen Daten liefert auch hier entsprechende PSNR-Werte, die in Abbildung 47 für eine Bildwiederholrate von 3 fps gezeigt sind. Bereits nach der zweiten Anfrage sind 98 der 100 Bilder komplett im Cache vorhanden. Der dritte Durchlauf ermöglicht das Auslesen der Bilder in originaler Qualität ohne Zugriff auf den Massenspeicher. Abbildung 48 zeigt die Messergebnisse bei einer Bildwiederholrate von 6 fps. Wie zu erwarten war, werden bei steigender Bildwiederholrate mehr Anfragerunden benötigt, bevor die komplette Datei im Cache zwischengespeichert ist. Nach fünf Runden konnte die angefragte Sequenz auch hier komplett aus dem Cache gelesen werden. Abbildung 49 zeigt eine Konfiguration mit 12 fps, bei der mit der verwendeten Testsequenz insgesamt 16 Runden benötigt wurden, bis die Daten komplett im Cache vorhanden waren.

5.4 Cache-Ersetzungsstrategien für skalierbare Medien

[20]Nachdem die Bilder entweder komplett oder teilweise vom Massenspeicher eingelesen und im Cache des Betriebssystems zwischengespeichert wurden, kommt es früher oder später zu einer Konkurrenzsituation bzgl. des zur Verfügung stehenden Cache-Speichers. In Abschnitt 2.2.4 wurden dateiorientierte Cache-Ersetzungsstrategien vorgestellt, die auf unterschiedliche Weise Kandidaten zur Verdrängung aus dem Speicher ermitteln, um Platz für neue Daten zu schaffen. Ein dateiorientiertes Vorgehen erscheint für nicht-skalierbare Bilder ebenfalls sinnvoll, da diese nur in ihrer Gesamtheit von aufrufenden Prozessen interpretiert und weiterverarbeitet werden können. Bei skalierbaren Medien hingegen kann neben dem dateiorientierten Ansatz auch ein paketorientiertes Konzept verfolgt werden, bei dem lediglich Skalierungsstufen der Daten verworfen werden, um freien Speicherplatz im Cache zu erhalten.

Im Besonderen sollen damit zwei wesentliche Werte reduziert werden: (i) Die Antwortzeit des Betriebssystems, da Anfragen — zumindest teilweise — aus dem Cache bedient werden können und (ii) die Cache-Miss-Rate der verwendeten Verdrängungsstrategie.

Ein simples Verwerfen beliebiger Pakete kann allerdings dazu führen, dass die restlichen Bilddaten keinen validen Codestream mehr bilden, da Pakete üblicherweise aufeinander aufbauen und somit eine gewisse Abhängigkeit innerhalb der Daten entsteht. Aus diesem

[20] Teilaspekte dieses Abschnitts wurden in [105] sowie [107] veröffentlicht.

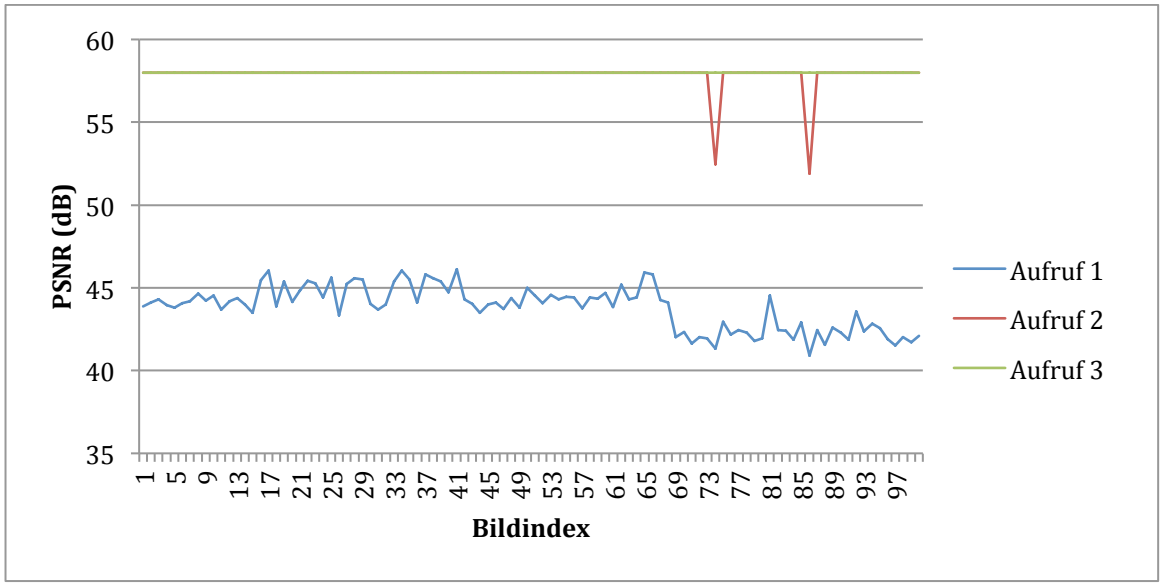

Abbildung 47: PSNR-Vergleich zwischen decodierter Originalsequenz und der unter Echtzeitbedingung gelieferten Variante vom virtuellen Dateisystem inkl. Caching und einer Bildwiederholrate von 3 fps (Progressionsreihenfolge: RLCP)

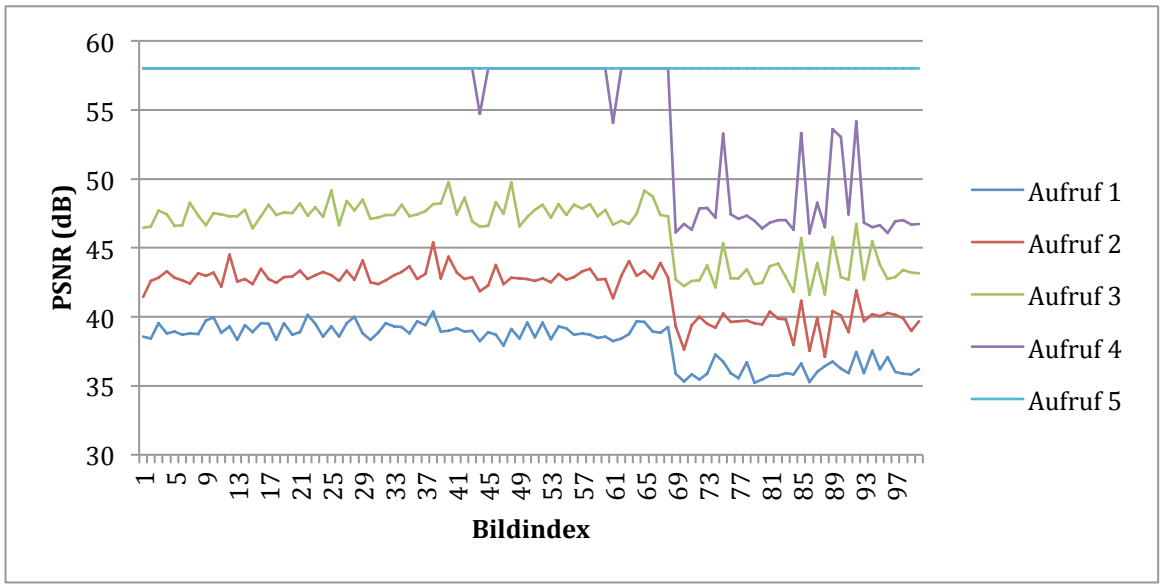

Abbildung 48: PSNR-Vergleich zwischen decodierter Originalsequenz und der unter Echtzeitbedingung gelieferten Variante vom virtuellen Dateisystem inkl. Caching und einer Bildwiederholrate von 6 fps (Progressionsreihenfolge: RLCP)

Grund wurden im Rahmen dieser Arbeit neue Verdrängungsstrategien (engl. *Replacement Algorithm* oder *Eviction Strategy*) für skalierbare Medien entwickelt, die im Besonderen die Möglichkeit ausnutzen, dass skalierbare Daten in ihrer Dateigröße verringert werden können und dennoch valide Daten enthalten, um z. B. ein Bild oder Video anzeigen zu können. Im Rahmen dieser Arbeit wurden die folgenden Cache-Verdrängungsstrategien untersucht:

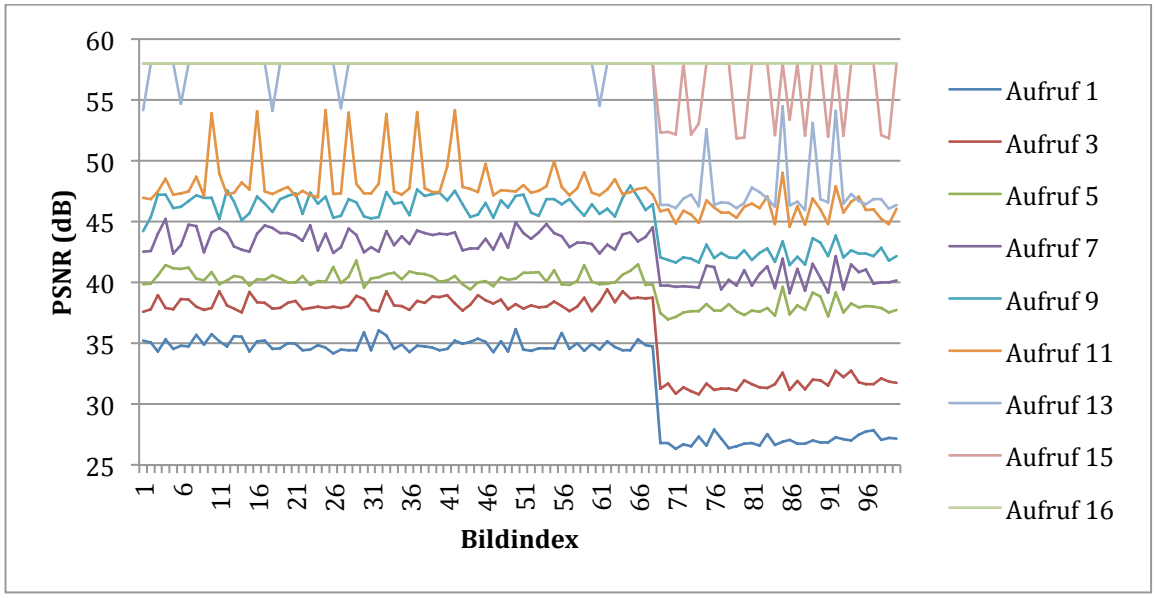

Abbildung 49: PSNR-Vergleich zwischen decodierter Originalsequenz und der unter Echtzeitbedingung gelieferten Variante vom virtuellen Dateisystem inkl. Caching und einer Bildwiederholrate von 12 fps (Progressionsreihenfolge: RLCP)

Verdrängung ungenutzter Pakete:

Beim Lesen vom Massenspeicher kann es aus verschiedenen Gründen dazu kommen, dass nicht benötigte Pakete eingelesen werden. Z. B. ist das bei Festplatten mit rotierenden Scheiben der Fall, wenn Datenpakete einer skalierbaren Datei sehr klein sind. Auf Grund der Latenz- und Spurwechselzeit (siehe auch 2.2.2) kann ein „Durchlesen“ schneller als eine Neupositionierung sein. Weiter kann es vorkommen, dass bereits eingelesene und zwischengespeicherte Pakete auf Grund von Zugriffsrechten nicht mehr von den aktuell angemeldeten Benutzern betrachtet werden dürfen. In diesen Fällen sind die ungenutzten Pakete ideale Kandidaten für eine Verdrängung, wenn der Cache vollläuft.

Verdrängung unvollständiger Pakete:

Wie in der echtzeitfähigen Einlesestrategie mit zusätzlichem Cache beschrieben, befinden sich üblicherweise Pakete im Cache, die nicht komplett gelesen werden konnten und somit nicht an aufrufende Prozesse weiter geleitet werden (vgl. Abbildung 46). Vielmehr werden diese Pakete mittels der Substitutionsmethode komplett durch ein Nullpaket identischer Länge ersetzt. Erreicht der Cache seinen maximalen Füllstand, sind unvollständige Pakete ebenfalls ideale Kandidaten für eine Verdrängung.

Verdrängung nach Skalierbarkeit:

Diese Strategie reduziert den besetzten Speicher einer skalierbaren Datei durch Verdrängung von Paketen, die zu einer bestimmten Skalierungsstufe gehören. Dies könnte bspw. die maximale Auflösungsstufe aller JPEG 2000-Bilder einer Sequenz sein, die sich aktuell im Cache befindet. In diesem Fall würde die ursprüngliche Auflösung (z. B. 4K) um eine

Auflösungsstufe (Level) reduziert werden. Im Fall von JPEG 2000 würde diese sowohl die horizontale wie auch vertikale Auflösung halbieren, wenn keine weitere Kompensation stattfindet. Allerdings würde die spontane Reduktion der Paketanzahl — und damit der eigentlichen Dateigröße — für einen aufrufenden Prozess nicht nachvollziehbar sein. Aus diesem Grund werden auch hier verdrängte Pakete bei einer erneuten Anfrage durch die Substitutionsmethode ersetzt. Hierdurch bleiben Größe und Struktur der Datei erhalten und aufrufende Prozesse müssen nicht auf spontane Änderungen reagieren. Neben der Auflösung bietet die Qualität eine weitere Möglichkeit, Pakete nach der Skalierbarkeit zu verdrängen. Die aktuelle Implementierung erlaubt dabei eine Mehrfachausführung einer beliebigen Reduktion. Diese kann z. B. dann erfolgen, wenn bereits zuvor eine erste Reduktion einer Skalierungsstufe — z. B. der Auflösung — stattgefunden hat, und anschließend freie Speicherbereiche durch das Einlesen neuer Bilder belegt wurden, so dass der Cache erneut seinen maximalen Füllstand erreicht. Dadurch wird es nötig, eine weitere Auflösungsstufe aller Bilder zu verwerfen. Selbstverständlich müssen die Bilder dazu genügend Auflösungsstufen beinhalten.

Verdrängung nach Paketgröße:

Als Alternative zur Verdrängung nach einer bestimmten Skalierungsstufe kann eine Verdrängung anhand der Paketgrößen erfolgen. Hierdurch wird in einigen zwischengespeicherten Bildern die höchste Auflösungsstufe verworfen, wohingegen bei anderen Bildern der gleichen Bildsequenz die höchste Qualitätsschicht verworfen wird. Ziel dieser Ersetzungsstrategie ist es, die maximale Menge an Speicherplatz freigeben zu können.

Verdrängung durch kombinierte Kriterien:

Diese Strategie erlaubt es bspw. zuerst die höchste Auflösungsstufe zu verwerfen. Erreicht der Cache danach erneut seinen maximalen Füllstand, z. B. weil neue Bilder dem Cache hinzugefügt wurden, wird in einem zweiten Ersetzungsschritt die höchste verfügbare Qualität verworfen usw.

5.4.1 Implementierung

Auf Basis des entwickelten Dateisystems für skalierbare Mediendaten (vgl. Abschnitt 4.1.1) wurden die vorgestellten Verdrängungsstrategien implementiert. Das Klassenmodell in Abbildung 50 gliedert dabei die implementierten Cache-Strategien in zwei Typen auf:

1. Paketbasierte Cache-Strategien (Abbildung 50, rechter Zweig)
2. Dateibasierte Cache-Strategien, wie bspw. FIFO und LRU (vgl. 2.2.4)

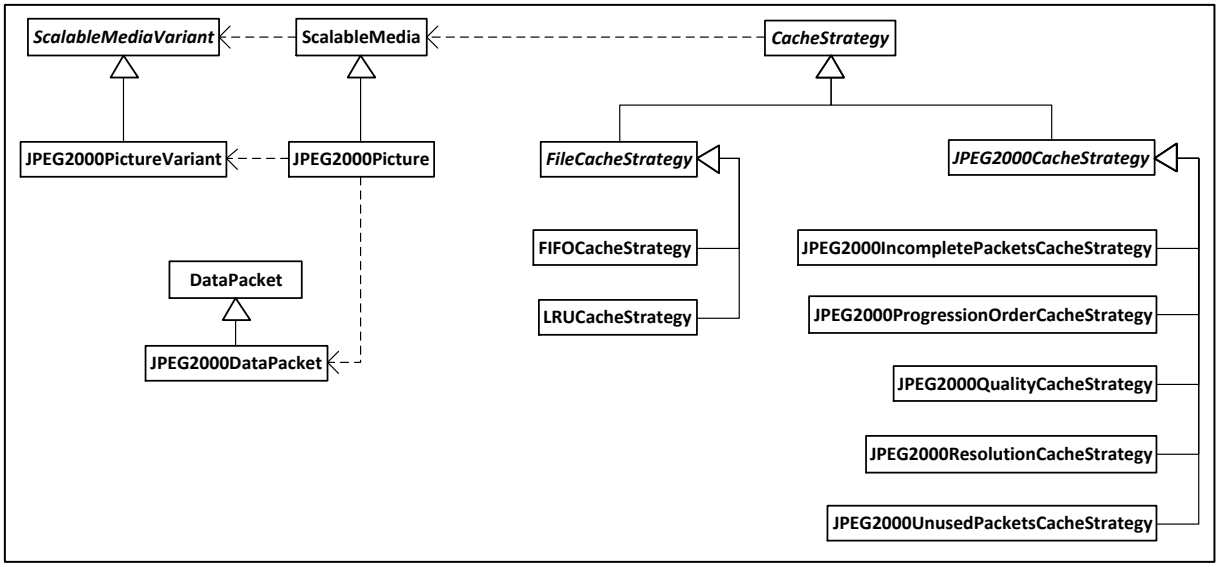

Abbildung 50: Klassenmodell zur Erweiterung des Dateisystems für skalierbare Medien um paketorientierte Cache-Verdrängungsstrategien

Für die restliche Anwendung ist die genaue Implementierung der Cache-Strategie abstrakt — es wird lediglich die abstrakte Klasse *CacheStrategy* verwendet. Da alle weiteren Cache-Strategien von dieser Klasse abgeleitet sind, können diese auch später wahlfrei ausgetauscht oder miteinander kombiniert werden, ohne dass Änderungen an der restlichen Applikation nötig werden. Eine mögliche Kombination wäre eine paketorientierte Verdrängungsstrategie gefolgt von einer dateiorientierten Strategie, wie z. B. FIFO. Hierbei würde zunächst die paketorientierte Strategie versuchen, bspw. Speicherplatz auf Kosten der Bildqualität durch Verwerfen einzelner Pakete der Bilder freizugeben. Dieser Vorgang lässt sich allerdings nicht beliebig häufig wiederholen, da entweder (i) die minimale Qualität der Bilder erreicht ist und keine weitere Reduktion mehr möglich ist oder (ii) die vom Dateisystem gewählte untere Qualitätsgrenze erreicht ist. Kommt es in diesem Fall zu einer weiteren Ausführung der paketbasierten Ersetzungsstrategie, wird kein Speicherplatz freigegeben. Aus diesem Grund werden die paketbasierten Verfahren mit dateiorientierten Verfahren kombiniert, da dadurch sichergestellt ist, dass stets Speicherplatz freigegeben wird. Eine nachgeschaltete FIFO-Strategie verwirft anschließend die verbliebenen Datenpakete aus dem Cache und gibt dadurch neuen Speicher frei.

Auf Grund des flexiblen Klassenmodells können viele weitere Kombinationen gewählt werden. Dabei bildet die Klasse *JPEG2000CacheStrategy* die Basisklasse für alle paketbasierten Verfahren, wohingegen *FileCacheStrategy* als Basisklasse für die dateiorientierten Verfahren dient. Durch Nutzung von Vererbung und virtuellen Basisklassen können weitere Cache-Strategien einfach hinzugefügt werden, ohne den Rest der Softwarearchitektur des Dateisystems für skalierbare Medien anpassen zu müssen.

Abbildung 51 zeigt die wesentlichen Schritte zur Erreichung des gewünschten Verhaltens bei der Verwendung paketorientierter Verdrängungsstrategien. Anfragen an Dateien, die das

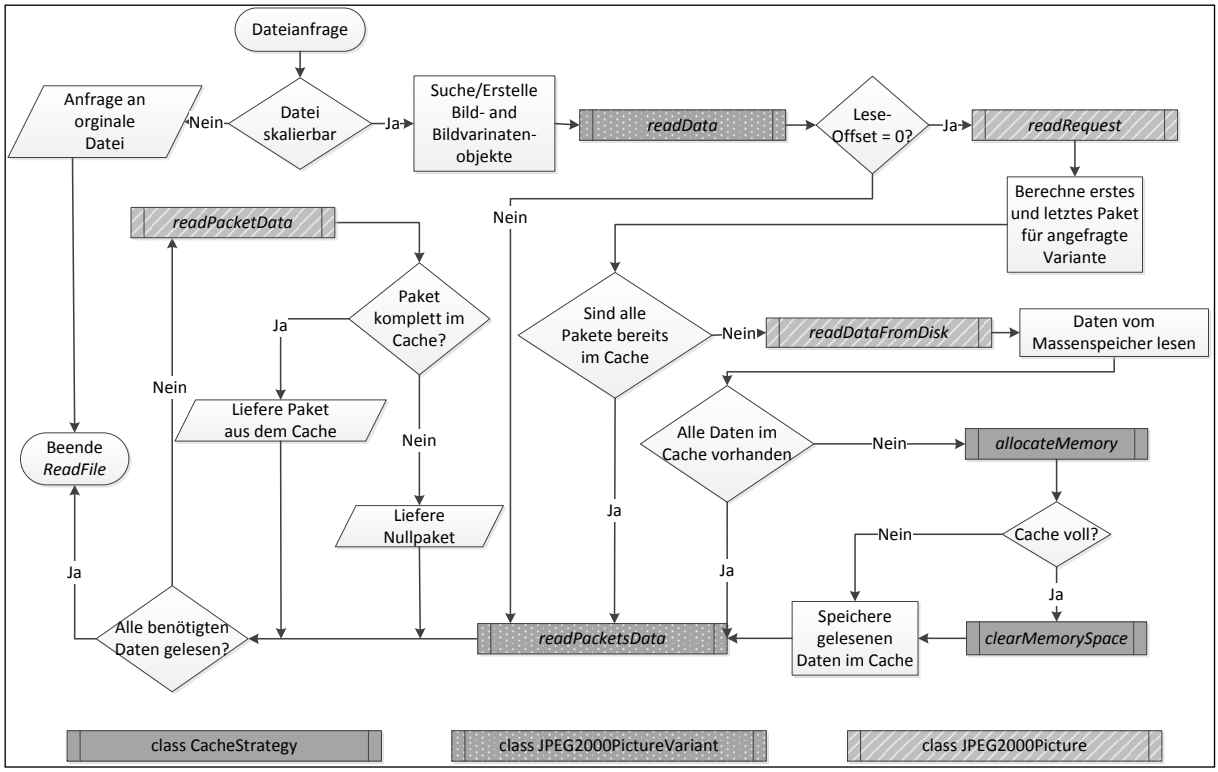

Abbildung 51: Prozessdiagramm zur Implementierung der nötigen Schritte für paketorientierte Cache-Verdrängungsstrategien

System als nicht-skalierbar einstuft, werden dabei direkt an das reale Dateisystem weitergegeben. Die gelieferten Daten vom realen Dateisystem werden nicht zwischengespeichert und ebenso direkt an den aufrufenden Prozess durchgereicht. Bei Leseanfragen zu skalierbaren Medien bestimmt der Algorithmus zunächst, ob bereits eine Variante der angefragten Datei im Cache vorhanden ist. Dies ist immer dann der Fall, wenn der Lese-Offset der Datei einen Wert größer Null beinhaltet. Ist bereits eine Variante vorhanden, wird an der Stelle weiter gelesen, an der beim letzten Lesezugriff aufgehört wurde. Ein Lesezugriff kann bspw. zur Einhaltung der Echtzeitfähigkeit (vgl. 5.1) unterbrochen werden.

Ist keine Variante vorhanden, werden die zu lesenden Bereiche der Variante bestimmt und damit begonnen, die Daten vom Massenspeicher zu lesen. Dabei überprüft der Algorithmus bei jedem Einlesen neuer Daten, ob noch genug Speicher im Cache vorhanden ist, um die Daten dort abzulegen. Ist das nicht der Fall, wird mit dem Aufruf der Funktion *clearMemorySpace* der Klasse *CacheStrategy* versucht, Speicher freizugeben. Nach dem Einlesen wird ermittelt, ob das komplette Bild im Cache vorhanden ist. Ist dies nicht der Fall, werden fehlende Pakete durch die Substitutionsmethode ersetzt und der resultierende Codestream an den aufrufenden Prozess zurückgeliefert.

5.4.2 Verhalten der Cache-Ersetzungsstrategien für skalierbare Medien

Um zu ermitteln, in welchem Maße die hier vorgestellten Verdrängungsstrategien dazu beitragen, die Reaktionszeit des Gesamtsystems zu verbessern, wurden zwei verschiedene Messungen durchgeführt:

1. Messungen bzgl. der Cache-Hit-Rate, bei denen ermittelt wurde, ob eine Variante der Datei im Cache vorrätig ist. Dabei wurde nicht berücksichtigt, in welcher Qualität bzw. Auflösung die Datei im Cache vorlag.

2. Messungen bzgl. der Signalqualität der verfügbaren Varianten im Cache durch den Vergleich des aktuellen Cachezustandes und der originalen Daten auf dem Massenspeicher. Auch in diesen Messreihen bedeutet ein PSNR-Wert nach (4.2) mit dem Wert 0, dass keine Variante im Zwischenspeicher vorliegt. Beim PSNR-Vergleich von zwei identischen Bildern ergibt sich ein Wert von ∞, der in diesen Messreihen ebenfalls durch einen frei gewählten Maximalwert von 58 dB ersetzt wurde. Wird dieser Wert erreicht, sind die zwischengespeicherten Varianten im Cache identisch zu den Varianten auf dem Massenspeicher. Alle Werte zwischen 0 dB und 58 dB zeigen, dass eine reduzierte Variante der skalierbaren Datei im Zwischenspeicher vorliegt.

5.4.3 Ergebnisse

Abbildung 52 und Abbildung 53 zeigen verschiedene Messungen für eine JPEG 2000-Sequenz mit 500 Bildern. Die verwendeten Bilder beinhalten fünf Auflösungs- sowie Qualitätsschichten, die durchschnittliche Dateigröße eines Bildes beträgt etwa 8,5 MB. Die komplette Sequenz wurde für die Messreihen einmal komplett vom virtuellen Dateisystem für skalierbare Medien angefragt. Dabei wurde die Cachegröße mit einem maximalen Wert von 100 MB festgelegt, wodurch üblicherweise maximal elf Bilder gespeichert werden können.

Abbildung 52 zeigt den Verlauf der Cache-Hit-Raten bei einer auflösungsorientierten Ersetzungsstrategie auf Basis von Paketen in Kombination mit einer dateiorientierten Ersetzungsstrategie für den Fall, dass die paketorientierte Strategie keinen Speicher mehr freimachen kann. Die Bilder werden demnach komplett aus dem Cache entfernt, wenn die Auflösung so häufig reduziert wurde, dass nur noch 50% (respektive 25% und 12%) der originalen Auflösung im Cache vorhanden sind. Es ist zu erkennen, dass um so mehr Bilder im Cache vorrätig sind, je niedriger das untere Limit der paketorientierten Cache-Ersetzungsstrategie gesetzt wurde. Bei einem unteren Limit von 12% befindet sich eine Variante aller Bilder der verwendeten Testsequenz im Cache. Wird der Minimalwert gesteigert, muss die Ersetzungsstrategie Bilder komplett entfernen. Hierzu wird die dateiorientierte Verdrängung verwendet. Dennoch befinden sich rund 100 Bilder im Cache, wenn das untere Auflösungslimit auf 50% gesetzt wird. Im Vergleich zur üblichen

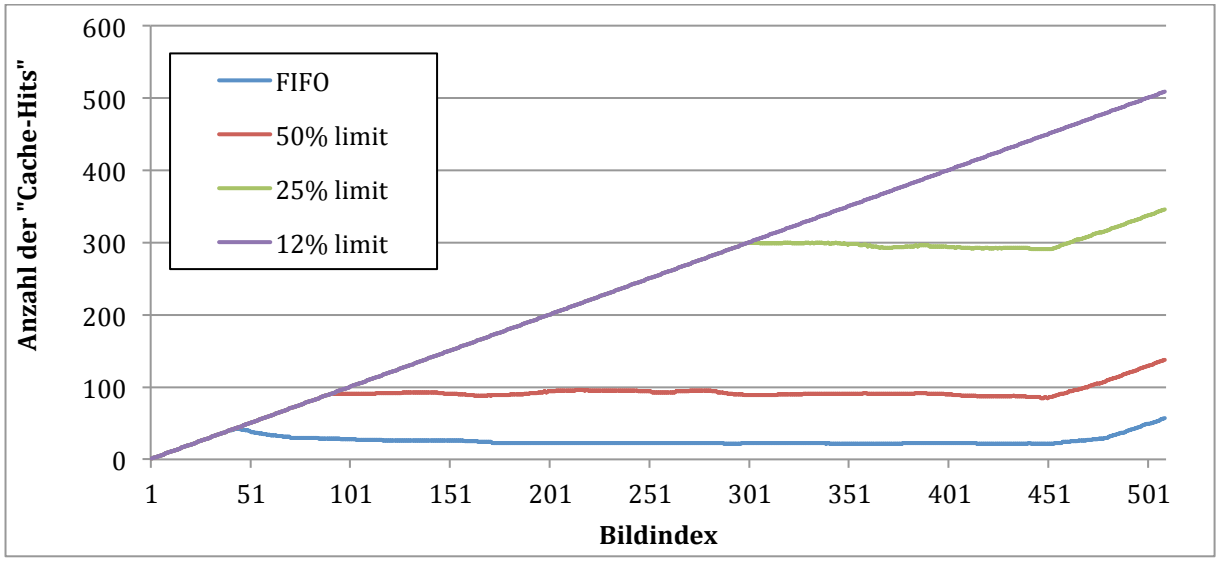

Abbildung 52: Cache-Hit Entwicklung unter Anwendung einer auflösungsorientierten Cache-Ersetzungsstrategie

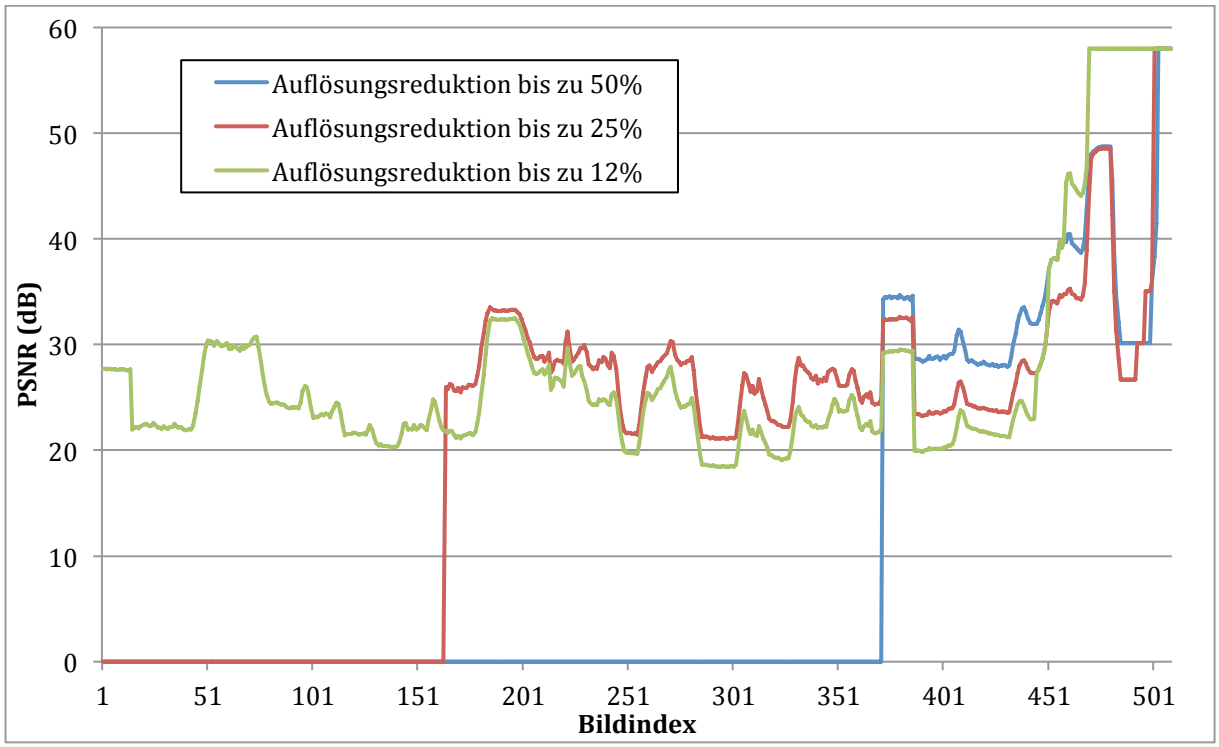

Abbildung 53: Ermittelte PSNR-Werte bei Anwendung einer auflösungsorientierten Verdrängungsstrategie der im Cache gespeicherten Varianten im Vergleich mit den originalen Bilddaten auf dem Massenspeicher

FIFO-Strategie kann die Cache-Hit-Rate etwa um den Faktor drei gesteigert werden. Auch wenn dieses Verhalten so zu erwarten war, zeigen die Ergebnisse, dass die Variation der unteren Grenze der paketorientierten Ersetzungsstrategie bei der verwendeten Testsequenz einen signifikanten Einfluss auf die Menge der vorgehaltenen Bilder im Cache hat. Demgegenüber zeigt Abbildung 53, dass der ermittelte PSNR-Wert nur wenig variiert, wenn die untere Grenze auf einen höheren Wert gesetzt wird. Auch wenn der Qualitätsaspekt ausschließlich auf die guten Kompressionseigenschaften von JPEG 2000 zurückzuführen ist, lassen die Ergebnisse darauf schließen, dass die vorgestellten Cache-Ersetzungsstrategien so

parametrisiert werden sollten, dass Bilder erst komplett aus dem Cache verworfen werden sollten, wenn ein sehr geringes, unteres Limit der verfügbaren Auflösung erreicht wurde.

Einen ähnlichen Trend zeigen Abbildung 54 und Abbildung 55 in denen eine qualitätsorientierte Verdrängungsstrategie auf Paketebene untersucht wurde. Im Vergleich zu den bisherigen Ergebnissen werden anstelle von Auflösungspaketen Qualitätspakete aus dem Cache entfernt, wenn dieser seinen maximalen Füllstand erreicht und neue Daten hinzugefügt werden sollen. Analog zu den vorherigen Messungen wurde auch hier das untere Limit der Ersetzungsstrategie variiert, um die Auswirkungen bei der verwendeten Testsequenz ermitteln zu können. Auch bei diesen Messungen steigerte sich erwartungsgemäß die Cache-Hit-Rate, je niedriger diese Grenze gewählt wurde. Bei der qualitätsorientierten Variante konnten allerdings nicht alle Bilder im Cache vorgehalten werden, auch wenn das untere Limit auf 20% gesetzt wurde. Im Vergleich zur FIFO- bzw. LRU-Strategie konnte die Cache-Hit-Rate aber dennoch um einen Faktor von zwei gesteigert werden. Die unterschiedlichen Auswirkungen lassen sich durch die Struktur der Testsequenz erklären. Dabei beansprucht bspw. die unterste Auflösungsvariante lediglich eine geringe Datenmenge, da die Bildgröße im Vergleich zur nächsthöheren Auflösungsstufe um 75% reduziert wird. Im Vergleich dazu kann die Größe der Qualitätspakete nahezu beliebig während des Codierungsvorgangs gewählt werden. Bei der verwendeten Testsequenz wurden dem untersten Qualitätspaket mehr Daten als dem untersten Auflösungspaket zugewiesen, wodurch die Abweichungen zwischen den Messungen zu erklären sind.

5.4.4 Bewertung

Die Anwendung von speziellen Cache-Ersetzungsstrategien für skalierbare Medien liefert eine solide Möglichkeit, um die Cache-Hit-Rate im Vergleich zu den Standardvarianten FIFO und LRU signifikant zu steigern. Diese Steigerung wird durch eine Reduktion der Bildqualität erreicht, die jedoch auf Grund der hochqualitativen Kompressionseffizienz heutiger Verfahren nicht sonderlich ins Gewicht fällt. Besonders im Bereich der digitalen Filmpostproduktion, bei der Bilder bspw. für Synchronisationsarbeiten häufig in einer Schleife betrachtet werden, können die vorgestellten Verfahren dazu beitragen, die Anzahl der übersprungenen Bilder (engl. *frame-drop*) zu reduzieren bzw. komplett zu eliminieren, wenn der Cache des Betriebssystems nicht ausreicht, um die komplette Bildsequenz in voller Auflösung und Qualität vorzuhalten. Vor allem wenn die maximale Auflösung und/oder Qualität nicht zwingend benötigt werden, können die vorgestellten Verfahren dazu beitragen, Echtzeitfähigkeit des Gesamtsystems zu gewährleisten. Dabei zeigte besonders die auflösungsorientierte Verdrängungsstrategie bei Anwendung mit einer JPEG 2000-Bildsequenz sehr gute Ergebnisse, sowohl beim Vergleich der PSNR-Werte mit den originalen Bildern, wie auch bei subjektiven, visuellen Tests auf einem 24-Zoll Monitor.

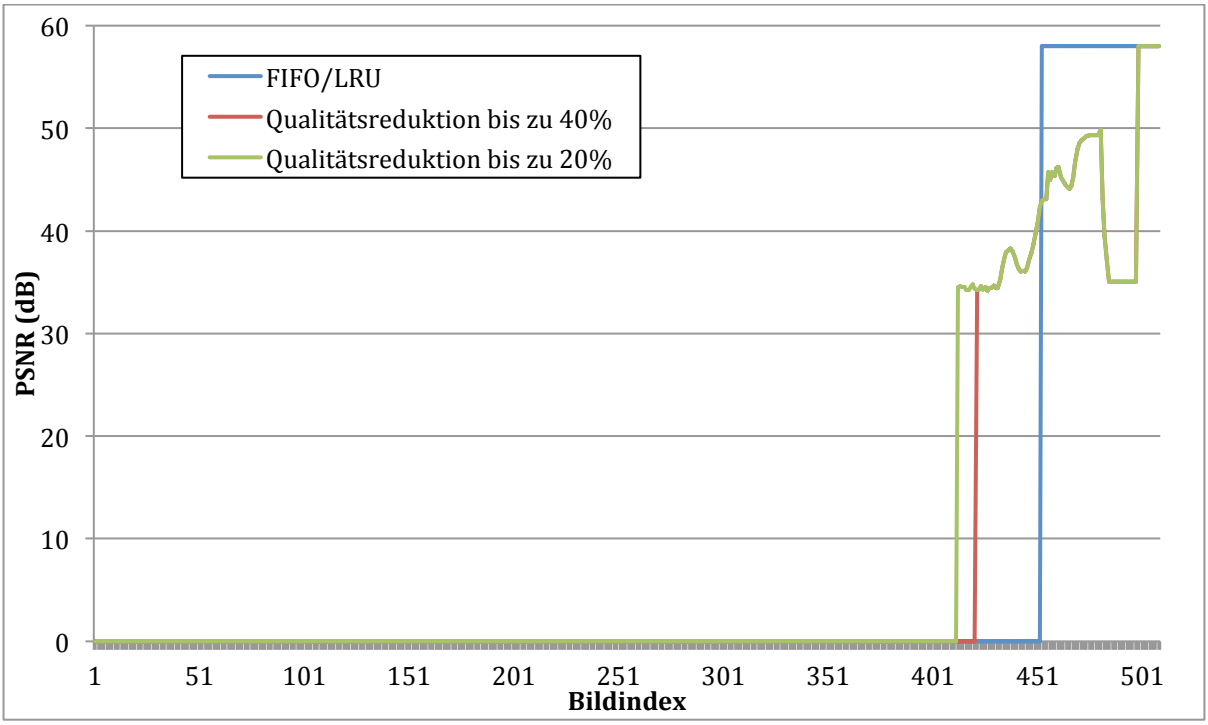

Abbildung 54: Ermittelte PSNR-Werte bei Anwendung einer qualitätsorientierten Verdrängungsstrategie der im Cache gespeicherten Varianten im Vergleich mit den originalen Bilddaten auf dem Massenspeicher

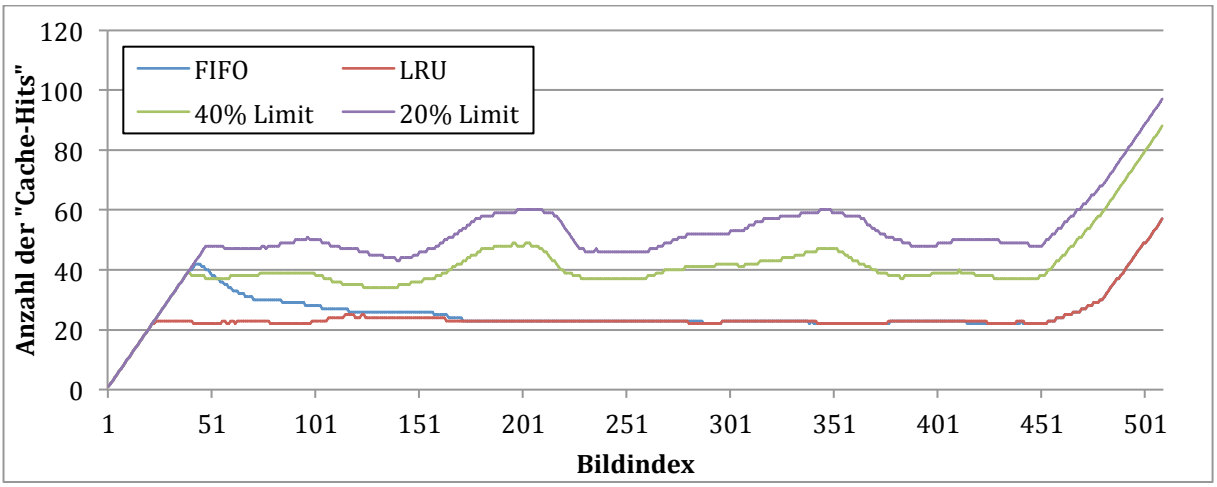

Abbildung 55: Cache-Hit-Entwicklung unter Anwendung einer qualitätsorientierten Ersetzungsstrategie

Die gezeigten Verfahren erfordern keine Änderungen in bestehenden Soft- und Hardwarelösungen, da eine Anwendung komplett innerhalb eines Dateisystems für skalierbare Medien stattfindet. Eine Änderung der Dateistruktur findet nicht statt, da verdrängte Pakete mit Hilfe der Substitutionsmethode (vgl. Kapitel 3) ersetzt werden. Die nichtlineare Steigerung der Rundenanzahl bei den vorgestellten Cache-Verfahren lässt sich mit dem konstanten Overhead des virtuellen Dateisystems begründen, der pro Aufruf bei etwa 31 ms liegt, unabhängig davon, wie viele Daten eingelesen werden. Dieser konstante Wert kommt bei geringen Bildwiederholraten, bei denen potenziell viele Daten pro Zugriff auf den realen Datenträger eingelesen werden können, weniger stark zum Tragen als bei hohen Bildwiederholraten.

6. Ergebnisse

Dieses Kapitel fasst die Ergebnisse aus den vorherigen Kapiteln zusammen und diskutiert die gewonnenen Erkenntnisse im Vergleich zum Stand der Technik. Ferner werden Impulse für weitere Forschungsarbeiten gegeben, die zu einer weiteren Akzeptanz von skalierbaren Kompressionsverfahren führen können.

6.1 Vergleich mit dem Stand der Technik

Der Vergleich zum Stand der Technik wird, entsprechend der vorherigen Kapitel, in verschiedene Bereiche untergliedert. Dabei zeigte sich, dass nur wenige Vorarbeiten für die Nutzung skalierbarer Kompressionsverfahren innerhalb von Computerarchitekturen vorhanden waren. Vielmehr gab es Vorarbeiten in verschiedenen Bereichen der Bildverarbeitung und Übertragung über Datennetze. Die Erkenntnisse der durchgeführten Arbeit werden nachfolgend aufgelistet.

1. Erweitertes Rechtemanagement

   Diese Arbeit nutzt zum ersten Mal den Ansatz einer Rechtevergabe auf Teile einer Datei, um dadurch verschiedene Varianten einer skalierbaren Mediendatei in Abhängigkeit der Zugriffsrechte anbieten zu können. Mittels der Implementierung eines virtuellen Dateisystems im Windows-Umfeld wurde ein Demonstrator erstellt, der die Funktionalität des gewünschten Rechtemanagements abbildet. Als besonderes Merkmal der Verfahren lässt sich herausstellen, dass die Generierung sowie Bereitstellung der virtuellen Dateien vollkommen transparent für den Benutzer sind.

   Ausgehend von den Herausforderungen bei der Verarbeitung von Mediendaten (vgl. 2.3) können zeitaufwendige Prozesse zur Erstellung diverser Proxy-Varianten im adressierten Einsatzgebiet durch Anwendung der vorgestellten Verfahren ersetzt werden. Neben der Ersparnis von Hardwareressourcen für das Erstellen der Proxies sowie des zusätzlichem Speicherplatzes für die neuen Varianten wird implizit der Aufwand zur Sicherstellung der Datenintegrität reduziert.

   Darüber hinaus diente der erstellte Prototyp als Grundlage für die weiteren Forschungsaktivitäten, die im Rahmen dieser Arbeit durchgeführt wurden.

2. Speicheradaptive Dateiorganisation

   Die Arbeiten von Kang et al. [53,128] zeigen Überlegungen, wie die Dateiablage skalierbarer Videoströme derartig durchgeführt werden kann, dass diese für eine spätere

Benutzung optimal ist und besonders der Lesedurchsatz von Festplatten mit rotierenden Scheiben und zusätzlicher Schreib-/ Lesekopfeinheit gesteigert werden kann. Bei diesen Verfahren gehen die Autoren davon aus, dass die Durchsatzrate während der Übertragung stark schwanken kann, wie es in einem Computernetzwerk der Fall sein kann. Im Gegensatz zu einem heterogenen Zugriffsverhalten, wie im Bereich von Multimedia-Servern, ist in der Filmpostproduktion mit einer homogenen Zugriffsart zu rechnen. Aus diesem Grund kann ein späteres Zugriffsverhalten sehr gut vorhergesagt werden. Als Erweiterung zu [53,128] zeigt die vorliegende Arbeit ein Konzept, das neben der Umstrukturierung von Bildsequenzen auch die Umstrukturierung auf Einzelbildebene erlaubt und dadurch weitere Durchsatzsteigerungen der verwendeten Datenträger erreichen kann. Darüber hinaus wird ein Konzept zur Neuorganisation der skalierbaren Daten vorgestellt, das zum Einsatz kommt, wenn sich das erwartete Zugriffsverhalten signifikant ändert und die vorher gewählte Ablagestrategie nicht mehr optimal für dieses Zugriffsverhalten ist.

Bei der Entwicklung der Verfahren standen die Aspekte zur Datenverarbeitung im Vordergrund, die innerhalb einer Computerarchitektur bei der Betrachtung von Einzelbildsequenzen relevant sind. Dabei galt das Ziel, möglichst viele Daten unter Echtzeitbedingung zu liefern, wobei ein sequenzielles Zugriffsverhalten der Anwender zu erwarten ist. Hierzu wurde ein Konzept vorgestellt, bei dem ein zu erwartendes Zugriffsmuster bereits bei der Speicherung der Daten auf einer klassischen Festplatte berücksichtigt wird. Die Ergebnisse zeigen, dass sich die Durchsatzraten von Festplatten mit rotierenden Bauteilen deutlich bis zu 400% steigern lassen, wenn die anwendungsfall-relevanten Bereiche der verwendeten Bilder möglichst sequenziell gelesen werden können.

Die vorgestellten RAID-Verfahren für skalierbare Medien liefern die ersten Ansätze, Dateien anhand ihrer internen Struktur auf verschiedene Datenträger zu verteilen. In dieser Arbeit konnte gezeigt werden, dass sich folgende Vorteile gegenüber der vorhandenen RAID-0-Strategie ergeben: (i) Durch die Berücksichtigung der Lesegeschwindigkeit beim Beschreiben der Datenträger kann sichergestellt werden, dass die ARAID-Konfiguration in Echtzeitanwendungen genutzt werden kann und Bildaussetzer auf Grund des Speichers nicht auftreten können. (ii) Bereits der erste Datenträger kann ohne Verfügbarkeit der anderen Datenträger für eine Vorschau verwendet werden. (iii) Im Falle eines defekten Datenträgers für Detailpakete, sind die Daten nicht komplett verloren, da auf dem ersten Datenträger noch eine komplett lesbare Subvariante zur Verfügung steht.

Ausgehend von den Herausforderungen bei der Verarbeitung von Mediendaten (vgl. 2.3) können zeitaufwendige Kopierprozesse im adressierten Einsatzgebiet durch Anwendung der vorgestellten Verfahren zukünftig verringert werden. Neben der Zeiter-

sparnis für das Umkopieren der Daten kann dadurch ebenfalls der Aufwand zur Sicherstellung der Datenintegrität mehrerer Kopien reduziert werden.

Wichtiger Bestandteil der Verfahren ist auch hier die Substitutionsmethode, die nicht zur Verfügung stehende Daten simuliert und dadurch ein valides Bild an die Decoder-Einheit liefert.

3. Echtzeitfähige Einlesestrategie

Die Nutzung der Skalierbarkeit während des Einlesens von einem Massenspeicher zeigt ein neues Anwendungskonzept der skalierbaren Medien. Der Stand der Technik konnte hier an verschiedenen Stellen erweitert werden. Das Konzept einer virtuellen Datei, die Anhand von Metadaten aus dem Header einer skalierbaren Bilddatei entnommen wird, konnte dabei aus [56] übernommen werden. Dabei wurden auch in dieser Arbeit die optionalen PL-Marker einer JPEG 2000-Datei verwendet, wie bereits in den Arbeiten [16] und [17] gezeigt wurde. Im Vergleich zu [43,114] müssen im Kontext dieser Arbeit jedoch nicht die Gegebenheiten der Übertragungsnetzwerke berücksichtigt werden. Zwar bietet das Konzept des JPIP-Servers geeignete Mechanismen bei der Anfrage skalierbarer Einzelbilder, wenn diese über Datennetzwerke verschickt werden. Für die Anwendung innerhalb einer Computerarchitektur und dem Einsatzgebiet der Filmproduktion zeigt die vorliegende Arbeit jedoch verbesserte Ansätze. Anstatt der Angabe eines *Window of Interest* (WoI) und einer gewünschten Qualität ist auch hier das Ziel, die maximale Datenmenge einer skalierbaren Datei einzulesen, die innerhalb einer vorgegebene Zeit geliefert werden kann. Demnach gibt die abgelaufene Zeit vor, wie viele Daten noch vom Datenträger eingelesen werden und nicht, wie in [43,114] vorgeschlagen, die bis dahin erreichte Qualität der decodierten Bilder. Eine Interaktion des Nutzers ist dabei nicht nötig.

Im Kontext dieser Arbeit galt es allerdings, der Decoder-Einheit stets eine standardkonforme Bilddatei zu liefern, unabhängig davon, wie viele Daten von der originalen Datei gelesen werden konnten. Im Stand der Technik wurde dies dadurch gelöst, indem eine Klient-/Server-Architektur die Verfahren der Skalierbarkeit vom restlichen System abstrakt hält und eine ständige Kommunikation zwischen den beiden Komponenten erfordert. Dieser Ansatz ist für die Nutzung innerhalb einer Computerarchitektur nicht nutzbar. Aus diesem Grund wurde anstelle einer proprioritären Kommunikation zum Austausch weiterer Daten der Ansatz gewählt, eine virtuelle Datei im Speicher nachzubilden und so lange mit realen Informationen zu füllen, wie Daten unter Angabe eines Zeitlimits vom Massenspeicher gelesen werden können. Kann ein Bild nicht komplett eingelesen werden, kommt es zu einer Kompensation der fehlenden Daten mittels der Substitutionsmethode. Das selbstentwickelte Verfahren ist daher geeignet, die Skalierbarkeit innerhalb der Computerarchitektur zu nutzen: Kommt es bei den bisher vorhandenen Verfahren zu einer Anpassung der kompletten

JPEG 2000-Datei, in Abhängigkeit der Datenmenge, die unter bestimmten Randbedingungen eingelesen werden konnte, bleiben Metadaten, Dateistruktur und Dateigröße bei den hier vorgestellten Verfahren konstant. Hierdurch erhält ein aufrufender Prozess genau die Datei, die dieser vom System angefragt hat.

Bei Mehrfachanfragen auf eine skalierbare Datei innerhalb einer Sitzung kann die Qualität der Medien durch jeden Aufruf gesteigert werden — ein Konzept, das im Rahmen von Datennetzwerken mit Proxy-Servern bereits in [89] vorgestellt und im Rahmen dieser Arbeit für die Anwendung innerhalb einer Computerarchitektur angepasst wurde. Analog zu [84] wird ein Abbild der bereits vom Massenspeicher eingelesenen Daten im Cache des virtuellen Dateisystems vorgehalten. Als Erweiterung muss auch in diesem Anwendungsfall dafür gesorgt werden, dass bei einer Anlieferung der Daten aus dem Cache eine standardkonforme Datei generiert wird, die in Struktur und Größe der originalen Datei vom Massenspeicher entspricht. Die hier erstellten Verfahren beruhen dabei auf einer zeitlichen Limitierung des Zugriffs auf die originale Datei.

Ausgehend von den Herausforderungen bei der Verarbeitung von Mediendaten (vgl. 2.3) können zeitaufwendige Umkopiervorgänge sowie der Einbau schnellerer Speicher- und Bussysteme im adressierten Einsatzgebiet durch Anwendung der vorgestellten Verfahren vermieden werden. Dies ist besonders für die Arbeitsschritte von Vorteil, bei denen eine Ausspielung in voller Qualität bzw. Auflösung nicht zwingend erforderlich ist. Beispielhaft soll hier der Schnittprozess innerhalb der Filmpostproduktion betrachtet werden. In diesem Arbeitsschritt werden die Bilddaten nicht direkt bearbeitet, sondern lediglich neue Metadaten in Form einer Schnittliste erstellt. Die volle Auflösung und Qualität der Bilddaten ist daher nicht entscheidend. Vielmehr ist eine Ausspielung ohne Bildaussetzer vorteilhaft, um die Güte der Übergänge an den Schnittgrenzen kontrollieren zu können. Andere Prozesse, wie bspw. die Farbkorrektur, würden von den vorgestellten Verfahren nicht profitieren, da hier eine Darstellung der maximalen Bildgröße bzw. Qualität zu bevorzugen ist und häufig nur ein Schlüsselbild (engl. *Key-Frame*) zur Festlegung der Farbwerte einer Szene benötigt wird.

Neben der Kompensation fehlender Durchsatzleistungen der heute weit verbreiteten Systeme sei darüber hinaus noch erwähnt, dass zukünftig bspw. eine Erhöhung der verwendeten Datenrate oder der Bildwiederholfrequenz dazu beitragen kann, dass auch leistungsstärkere Schnittstellen (z. B. Thunderbolt) an ihre Leistungsgrenzen stoßen werden. Auch hier bieten die vorgestellten Verfahren eine mögliche Kompensationsstrategie, wenn skalierbare Medien genutzt werden.

4. Parametrisierbare Dateiaufrufe

Im Gegensatz zum vorhandenen JPIP-Verfahren [43,114], sowie den Verfahren die mit dem Austausch einer Indexdatei arbeiten [16,17,56] zeigt diese Arbeit ein Kon-

zept, bei der Anfragen zu einer gewünschten Variante direkt durch den Dateiaufruf übermittelt werden können. Durch die optionalen Zusatzparameter kann ein aufrufender Prozess bestimmen, welche Eigenschaften die vom Dateisystem gelieferten Dateien aufweisen. Dies setzt jedoch voraus, dass die verwalteten Dateien entsprechende Skalierungen erlauben. Hierdurch erhält das Dateisystem die Aufgabe, die Bilder bei Bedarf in einer entsprechenden Subvariante anzuliefern. Der Stand der Technik kennt hierzu bereits die Klient-/Server-Architekturen, die u. a. in [43,114] gezeigt werden. Die Arbeiten zeigen ein Konzept, bei denen der Benutzer angibt, welcher Bereich eines Bildes betrachtet werden soll. Der Server entscheidet, je nach aktueller Auslastung, welche Daten der Klient für das angefragte WoI erhält. Vorteilhaft zeigt sich, dass das aufrufende Programm keine Kenntnis über die Struktur der skalierbaren Daten aufweisen muss, sondern abstrakte Formulierungen an einen weiteren Prozess absetzen kann, der über entsprechende Kenntnisse verfügt. Dieses Konzept wurde für den parametrierbaren Dateiaufruf übernommen. Es erfolgt die Anfrage an das Dateisystem, das in diesem Fall die Skalierung abstrakt hält. Vor allem wurden im Rahmen dieser Arbeit die Verarbeitungsschritte innerhalb einer Computerarchitektur berücksichtigt. Dabei ist entscheidend, dass das Dateisystem bereits bei der Anfrage der Dateigröße einer Subvariante eine verlässliche Aussage liefert, wie groß die skalierte Datei anschließend sein wird. Identische Parametrierung vorausgesetzt, kann ein aufrufender Prozess danach entsprechend den benötigten Speicherplatz allokieren und die Datei einlesen. Nicht zuletzt deshalb bleiben Anfragen, im Gegensatz zu [43,114], unverändert und der Benutzer erhält die gewünschte Datei, sofern diese als Subvariante extrahierbar ist.

Somit zeigt die vorliegende Arbeit die erste Realisierung eines Dateisystems mit der Möglichkeit, Seiteninformationen in den Dateiaufruf einzubetten, um dadurch die Darstellungsform einer (skalierbaren) Multimediadatei anzupassen. Eine Implementierung zur Nutzung der Skalierbarkeit ist dadurch nicht mehr zwangsläufig auf der Applikationsebene nötig.

5. Cache-Verdrängungsstrategien für skalierbare Medien

Besonders im Web-Umfeld existieren viele Vorarbeiten auf dem Gebiet der Cache-Verfahren für skalierbare Medien. Eine Anwendung von Cache-Einlagerungs- und Cache-Ersetzungsstrategien für skalierbare Medien innerhalb von Computerarchitekturen wurde bisher nicht untersucht. Diese Lücke wird durch die vorliegende Arbeit geschlossen. Dabei konnten verschiedene Verfahren aus dem Stand der Technik übernommen, verbessert und auf das adressierte Einsatzgebiet angepasst werden. Das Konzept aus [89], bei der die Qualität der angefragten Medien sukzessive steigt, wenn diese mehrmals angefragt werden, wurde in der vorliegenden Arbeit adaptiert. Dabei muss gewährleistet sein, dass die bereits vorher eingelesenen Daten nicht schon wieder aus dem Cache entfernt wurden. Ein Nachladen von Daten vom Dateiserver in

[89] entspricht einem Nachladen weiterer Pakete vom verwendeten Datenträger. Ohne eine Klient-/Server-Architektur funktioniert ein solches Verfahren nicht ohne Weiteres in einem Computersystem, da aufrufende Programme keine Kenntnis darüber besitzen, dass bspw. noch nicht alle Daten einer Datei zur Verfügung stehen und somit davon ausgehen, dass sie auf eine valide Datei zugreifen. Ein zusätzliches Laden der fehlenden Daten ist dabei auf Grund des Wunsches nach einer unterbrechungsfreien Ausspielung unvorteilhaft, weshalb in dieser Arbeit eine Kombination mit der Substitutionsmethode vorgestellt wird. Erst hierdurch wird die Anwendung im adressierten Einsatzbereich möglich, da das Betriebssystem jederzeit eine valide Datei an den aufrufenden Prozess liefern kann.

Auch im Bereich der Cache-Verdrängungsstrategien existieren zahlreiche Vorarbeiten. Dabei finden Verfahren, die auf Messung und Auswertung der Zugriffshäufigkeit auf bestimmte Pakete/Schichten beruhen [29,51,63,83,111], nur bedingt Anwendung innerhalb der vorliegenden Arbeit. Der Grund dafür ist, dass bei einer Echtzeitausspielung üblicherweise alle Pakete eines skalierbaren Bildes die gleiche Zugriffshäufigkeit aufweisen, da der Benutzer die Daten meist in der maximal erreichbaren Qualität/Auflösung angezeigt bekommen möchte. Da fehlende Daten bereits mit der Substitutionsmethode kompensiert werden müssen, ist üblicherweise nicht davon auszugehen, dass Pakete auf Grund von dedizierten Nutzeranforderungen nicht gelesen werden sollen und sich dadurch unterschiedliche Zugriffsfrequenzen für einzelne Pakete/Schichten ergeben. Dennoch ist dieser Aspekt bei konkurrierenden Aufrufen mehrerer Nutzer denkbar — eine Erweiterung der gezeigten Verfahren in Sinne der Zugriffsfrequenz auf bestimmte Datenpakete der skalierbaren Daten deshalb eine potenzielle Erweiterungsmöglichkeit für nachfolgende Forschungsarbeiten. Weiter stellt das größenadaptive Verfahren aus [9] eine mögliche Erweiterung dar. Anwendung in der vorliegenden Arbeit kann derart erfolgen, dass lediglich eine bestimmte Subvariante im Cache eingelagert wird, wenn das System sicherstellen kann, dass die fehlenden Pakete für die Variante mit maximaler Qualität bzw. Auflösung unter Echtzeitanwendung vom potenziell langsamen Datenträger nachgeladen werden können.

Die Erkenntnisse von Zink et al. bzgl. der Wahrnehmung beim Betrachter [134,135,136] wurden in der vorliegenden Arbeit derart abgebildet, dass die (teilweise) zwischengespeicherten Einzelbilder im gleichen Maße in einer Skalierungsstufe reduziert werden, wenn der Cache seinen maximalen Füllstand erreicht. Hierdurch wird gewährleistet, dass Schwankungen in der wahrgenommenen Qualität gering bleiben.

Ausgehend von den Herausforderungen bei der Verarbeitung von Mediendaten (vgl. 2.3) können durch Anwendung der vorgestellten Cache-Verfahren, in Abhängigkeit der verwendeten Bildsequenz sowie der gewählten, paketorientierten Verdrängungsstrategie, mehr Bilder im Cache vorgehalten werden, als es im Vergleich zu klassi-

schen Verdrängungsstrategien möglich ist. Hierdurch kann eine angestrebte Ausspielung ohne Bildaussetzer unterstützt werden. Die Anwendung der vorgestellten Strategien ist dabei nicht für jeden Prozess im adressierten Einsatzgebiet vorteilhaft, unterstützt jedoch die Arbeitsschritte, bei denen die Verarbeitung der maximalen Qualität bzw. Auflösung der Bilder nicht oberstes Ziel ist.

Zusammenfassend lässt sich sagen, dass diese Arbeit folgende neue Konzepte zeigt. Dieses sind vor allem die Verfahren zur Rechteverwaltung, die echtzeitfähige Einlesestrategie sowie die Verfahren für die parametrisierbaren Dateiaufrufe. Die Verfahren zur speicheradaptiven Dateiorganisation sowie Cache-Verdrängungsstrategien wurden zwar schon in anderen Bereichen der Bildverarbeitung teilweise adressiert. Bei den Vorarbeiten wurden die beiden wesentlichen Aspekte dieser Arbeit, die gewünschte Echtzeitfähigkeit sowie die Nutzung innerhalb einer Computerarchitektur mit *Handle*-basierten Zugriffen auf Ressourcen, nicht evaluiert. In dieser Arbeit wird diese Lücke geschlossen und gezeigt, dass die Nutzung von skalierbaren Medien bestimmte Bearbeitungsprozesse signifikant beschleunigen kann.

Als zentraler Prozess hat sich dabei die vorgestellte Substitutionsmethode herausgestellt, die dafür sorgt, dass die interne Struktur der skalierbaren Dateien — sowie deren Größe — konstant bleibt. Erst hierdurch wird die Nutzung der Skalierbarkeit innerhalb einer Computerarchitektur möglich.

6.2 Diskussion und Ausblick

Die vorliegende Arbeit beschreibt eine Sammlung neuer Verfahren zur Ausnutzung der Skalierbarkeit von Mediendaten innerhalb einer Computerarchitektur am Beispiel von Herausforderungen im Bereich der professionellen und semiprofessionellen Filmproduktion sowie der Distribution digitaler Filmpakete. Dabei werden sowohl neue Verfahren vorgestellt, wie auch bekannte Verfahren auf die besonderen Gegebenheiten der Verarbeitungskette innerhalb einer Computerarchitektur für das o. g. Einsatzgebiet angepasst. Das Hauptaugenmerk liegt dabei auf Verfahren, die eine Echtzeitausspielung der skalierbaren Daten auch dann ermöglicht, wenn verschiedene Komponenten — wie z. B. der verwendete Massenspeicher oder die Schnittstelle zwischen Massenspeicher und Computer — nicht hinreichend durchsatzstark für die komplette Datenmenge sind. Im Gegensatz zu den zahlreichen Vorarbeiten auf dem Gebiet der Anwendung skalierbarer Mediendaten, die vorrangig im Web-Umfeld zur Kompensation schwankender Durchsatzleistung der Datennetzwerke angesiedelt sind, müssen nicht lieferbare Daten in diesem Szenario simuliert werden, da aufrufende Prozesse nicht auf spontane Größen- sowie Strukturänderungen angefragter Dateien reagieren können. Mittels der hier vorgestellten Substitutionsmethode wurde die dazu fehlende Komponente geliefert.

Die durchgeführten Evaluationstests implizieren, dass eine Ausnutzung der Skalierbarkeit innerhalb einer Computerarchitektur mit vielen Vorteilen einhergeht. Hier ist vor allem die Echtzeitfähigkeit zu nennen, die in verschiedenen Bereichen der Filmproduktion wichtiger als die Darstellung der maximalen Auflösung oder Qualität ist. Kritiker mögen an dieser Stelle bemerken, dass SSD-Festplatten bereits heute in der Lage sind, Einzelbildsequenzen für die Kino- und Archivanwendung in Echtzeit zu liefern. Neben den hohen Kosten, die während der Durchführung dieser Arbeit noch für SSD-Speicher aufgerufen wurden, muss allerdings bemerkt werden, dass verschiedene Trends in der Filmproduktion dazu führen, dass auch SSD-Festplatten zu langsam für eine Echtzeitanlieferung der Daten werden. Zum einen sind hier die klassischen Parameter wie Datenrate, Bildwiederholrate und Auflösung zu nennen, die erfahrungsgemäß immer steigen, wenn entsprechend neue oder modifizierte Speichertechnologien und Netzwerkdurchsätze die Anwendung erlauben. So wurden bereits erste Kinofilme gezeigt, die mit einer Datenrate von 500 MBit/s erstellt wurden — höhere Datenraten sind bereits im Gespräch. Auch auf der Produktionsseite geben moderne Aufnahmeverfahren, mit 16 oder mehr parallel aufnehmenden Kameras, Anlass dazu, über die Nutzung von skalierbaren Codecs nachzudenken. Damit werden heute verfügbare Systeme in wenigen Jahren nicht mehr ausreichen, um die dann anfallenden Datenmengen in Echtzeit liefern zu können. Zum anderen unterliegen viele Prozesse bei der Distribution von Kinofilmen einem derart hohen Kostendruck, dass eine Anwendung der neuesten (und meistens kostenintensivsten) Technologie selten möglich ist. Da die vorgestellten Verfahren in Software abgebildet werden können, stellen sie vielversprechende Ansätze dar, auf der einen Seite vorhandene Produktionssysteme zu verbessern. Auf der anderen Seite ist es die Hoffnung des Autors, dass durch diese Arbeit gezeigt werden kann, welche Vorteile die Nutzung der Skalierbarkeit von Mediendaten neben den bekannten Anwendungen im Web-Umfeld noch haben kann. Somit kann diese Arbeit hoffentlich einen Beitrag leisten, die Grundidee der Skalierbarkeit attraktiver und bekannter zu machen.

Besonders sollen die erzielten Ergebnisse und Erkenntnisse zeigen, dass signifikante Leistungssteigerungen zu erzielen sind, ohne dass dafür hohe Kosten in Bezug auf Rechenressourcen oder Speicherkapazitäten entstehen. Vielmehr soll die vorliegende Arbeit zeigen, dass häufig nur minimale Änderungen in der Verarbeitungskette nötig sind, um signifikante Verbesserungen zu erzielen.

Bei der Durchführung sind viele Fragen aufgetaucht, die nicht im Rahmen des aktuellen Forschungsvorhabens beantwortet werden konnten. Die folgende Auflistung zeigt dabei einige Punkte, in welche Richtung die Entwicklung im adressierten Einsatzgebiet nach Meinung des Autors noch gehen kann, ohne dabei Anspruch auf Vollständigkeit zu erheben:

- Verbesserung der Substitutionsmethode von einer grobkörnigen (*coarse-grain*) zu einer feinkörnigen (*fine-grain*) Ersetzungsstrategie. Ziel dabei ist es, unvollständige Pakete nicht komplett mit Nullpaketen zu ersetzen. Vielmehr beinhaltet jedes Codestream-Paket viele sog. *Codepässe*, bei den jeder Codepass weitere Details zu

dem bereits decodierten Daten hinzufügt. Durch eine Erweiterung der bestehenden Substitutionsmethode könnten zukünftig alle kompletten Code-Pässe mit in die virtuelle Datei geschrieben werden. Es ist zu erwarten, dass die zeitliche Performanz der Substitutionsmethode hierdurch etwas sinkt, die Bildqualität hingegen zunimmt. Besonders bei sehr großen Paketen, die nur in mehreren Iterationen gelesen werden können, würden sich bei einer feingranularen Ersetzungsstrategie nach jedem Zugriff kleinere Verbesserungen der Bildqualität einstellen, wohingegen bei der heutigen Variante über mehrere Iterationen keine Änderung auftritt, bevor dann in einem Schritt eine große Qualitätsverbesserung erreicht wird.

- Die Implementierung des Dateisystems für skalierbare Medien, das im Rahmen dieser Arbeit vorgestellt wurde, erfolgte bewusst mit Hilfe eines virtuellen Dateisystems. Wie gezeigt wurde, ist die Leistung vom virtuellen Dateisystem in Bezug auf Datendurchsatz schlechter im Vergleich zu einem realen Dateisystem. Aus diesem Grund sollten die vorgestellten Verfahren in zukünftigen Arbeiten in ein reales Dateisystem überführt bzw. in ein vorhandenes Dateisystem integriert werden, um neben des erweiterten Funktionsumfanges auch Nutzen aus der höheren Durchsatzleistung ziehen zu können, die von einem Dateisystem zu erwarten ist, das vollständig im Kernel-Modus des Betriebssystems ausgeführt wird.

- Die vorgestellten Verfahren und deren Evaluierung erfolgten ausschließlich mit dem Einzelbild-Codec JPEG 2000. Die Evaluation weiterer skalierbarer Kompressionsformate — sowohl für Video, Einzelbilder und Audioformate — sollte Bestandteil weiterer Forschungsarbeiten sein.

- Die vorgestellten Echtzeitverfahren berücksichtigen bisher noch keine konkurrierenden Zugriffe auf limitierende Ressourcen, die durch den parallelen Zugriff mehrerer Prozesse auf unterschiedliche Daten auftreten. Konkurrierende Zugriffe sind allerdings alltäglich und sollten in zukünftigen Arbeiten genauer betrachtet werden. Eine Erweiterung der Cacheverdrängungsstrategien um Verfahren, die auf die Zugriffsfrequenz bestimmter Bereiche einer skalierbaren Datei beruhen [29,51,63,83,111], kann dabei ebenfalls realisiert werden.

- Ein vorrausschauendes Lesen der Daten bei Übertragungspausen, wie u. a. in [13,14,15,69,70] vorgestellt, findet bisher noch keine Anwendung innerhalb der entwickelten Einlesestrategie. Die Erweiterung um eine derartige Funktionalität wäre denkbar, da eine Vorhersage zum mutmaßlichen nächsten angefragten Bild besonders bei Videosequenzen trivial ist.

7. Zusammenfassung

Die vorliegende Arbeit leistet einen Beitrag zum Forschungsgebiet der Datenverarbeitung und Datenverwaltung innerhalb von Computersystemen unter Nutzung skalierbarer Dateiformate. Sie wird motiviert durch die Erkenntnis, dass skalierbare Dateiformate, im speziellen komprimierte Audio- und Videodaten, auf der einen Seite nicht gesondert durch vorhandene Datei- und Betriebssysteme unterstützt werden, auf der anderen Seite eine Nutzung dieser Formate jedoch zu verschiedenen Problemlösungen beitragen kann. Der Fokus dieser Arbeit liegt auf aktuellen Problemstellungen im Bereich der professionellen und semiprofessionellen Filmproduktion sowie der Filmdistribution von digitalen Filmpaketen, bspw. für Kino- und Archivanwendungen. Hierzu werden die für diese Arbeit relevanten, skalierbaren Kompressionsformate betrachtet und erläutert, wie die Skalierbarkeit verfügbarer Multimediaformate erreicht wird.

Die Forschungsarbeit wird durch die systematische Analyse von Einflussfaktoren auf die Durchsatzleistung in einem Computersystem begonnen. Hierzu werden die aktuell verfügbaren Systemkomponenten — darunter relevante Schnittstellen, Massenspeicher, Dateisysteme und Speicherkonzepte eines Computers — betrachtet und die Limitierungen bei der Verarbeitung nicht-skalierbarer Multimediadaten identifiziert und beschrieben, wie diese Engpässe mit heute verfügbaren Techniken meist umständlich umgangen werden. Ausgehend von diesen Problemstellungen werden Lösungsansätze abgeleitet, durch die entstehende Engpässe unter Nutzung skalierbarer Medien kompensiert werden können, ohne dass hierzu Zwischenverarbeitungsschritte notwendig sind.

Grundlegender Algorithmus für die Anwendbarkeit verschiedener Verfahren, die nachfolgend vorgestellt werden, ist die Einführung der sog. *Substitutionsmethode*. Sie erlaubt es, fehlende Daten eines skalierbaren Datensatzes — die bspw. auf Grund mangelnder Durchsatzleistung nicht unter Echtzeitbedingung eingelesen werden können — derart zu simulieren, dass nach der Ersetzung der fehlenden Daten wieder ein standardkonformer Datensatz vorliegt, der an die nachfolgenden Komponenten weitergeleitet werden kann. Die Simulation fehlender Daten ist notwendig, da innerhalb einer Computerarchitektur — im Gegensatz zu einer Client/Server-Architektur — nicht signalisiert werden kann, dass nicht alle Daten eines Datensatzes geliefert werden können. Eine mögliche Folge wären nicht vorhersehbare Zugriffe auf unbelegten Speicher, wenn die fehlenden Daten nicht simuliert werden. Die substituierten Datensätze stellen valide Strukturen bereit, sind aber auf Grund der simulierten Informationen in ihrer Qualität oder Auflösung im Vergleich zum originalen Datensatz vermindert.

Neben der Einführung der Substitutionsmethode lassen sich die Optimierungen vorhandener Arbeitsabläufe unter Nutzung skalierbarer Medien in zwei Bereiche zusammenfassen:

1. Konzepte zur Verwaltung skalierbarer Mediendaten – in diesem Bereich wird ein virtuelles Dateisystem konzipiert, das per Design Kenntnis über die Struktur und den internen Aufbau der relevanten, skalierbaren Formate hat. Zugriffsrechte lassen sich hierdurch auf bestimmte Bereiche einer Datei vergeben, wodurch verschiedenen Benutzern Zugriff auf verschiedene Varianten der skalierbaren Medien gewährt werden kann. Hierdurch entsteht im Rahmen dieser Arbeit das erste Dateisystem für skalierbare Mediendaten, was die Problemstellung der sog. Proxy-Generierung — bei der verschiedene Varianten einer Mediendatei erstellt werden, um dadurch die Benutzerrechte verschiedener Akteure abzubilden — in heutigen Abläufen in der Filmproduktion beseitigen kann. Weiter werden in diesem Bereich speicheradaptive Dateiorganisationen vorgestellt, die sowohl die Eigenschaften bestimmter Speichermedien, wie Festplatten mit rotierenden Scheiben oder Bandlaufwerke, ebenso berücksichtigen wie die Struktur skalierbarer Medien. Die vorgestellten Verfahren erlauben dabei eine Durchsatzsteigerung der verwendeten Speichersysteme wie auch die Sicherstellung der Echtzeitfähigkeit bei Verwendung von Bandspeicherlaufwerken in Kombination mit der bereits eingeführten Substitutionsmethode. Hierdurch müssen skalierbare Multimediadaten zukünftig nicht mehr von potenziell langsamen Bandlaufwerken auf schnellere Medien umkopiert werden. Dies ist besonders nützlich und zeitsparend, wenn die Daten lediglich für eine kurze Vorschau gesichtet werden müssen und die maximale Auflösung bzw. Qualität nicht zwangsläufig benötigt wird.

2. Verbesserung der Verarbeitungsabläufe unter Ausnutzung der Skalierbarkeit – in diesem Bereich werden bekannte Arbeitsabläufe unter Nutzung nicht-skalierbarer Medien untersucht und durch Nutzung skalierbarer Medien verbessert. Zunächst wird eine echtzeitfähige Einlesestrategie vorgestellt, die bei Bedarf einen Einlesevorgang von einem potenziell langsamen Datenträger — bzw. einem schnellen Datenträger, der über eine langsame Schnittstelle angebunden ist — abbricht, um eine vorgegebene maximale Einlesezeit nicht zu überschreiten. Fehlende Daten werden anschließend mit der Substitutionsmethode kompensiert. Hierdurch lässt sich die maximale Dauer für einen Einlesevorgang bestimmen und eine Antwort für Echtzeitwiedergabe erzwingen. Durch dieses Verfahren lassen sich ebenfalls Umkopiervorgänge in der Filmproduktion reduzieren, die auf Grund von langsamen Speichermedien bzw. langsamen Schnittstellen durchgeführt werden. Weiter wird in diesem Bereich das Konzept der parametrisierbaren Dateiaufrufe vorgestellt, die es erlauben, eine bestimmte Variante einer skalierbaren Datei vom Dateisystem anzufragen. Hierdurch müssen nachgeschaltete Hard- und Softwarekomponenten nicht zwangsläufig über die Kenntnis zur Verarbeitung skalierbarer Medien verfügen – was besonders im Fall von Hardware-Decodern für eine Abwärtskompatibilität sorgt. Abschließend werden Cache-Einlagerungs- so-

wie Cache-Ersetzungsstrategien vorgestellt, welche die Struktur skalierbarer Medien berücksichtigen und auf Basis des bereits vorgestellten, virtuellen Dateisystems implementiert werden. Entsprechende Messreihen zeigen dabei, dass die sog. *Cache-Hit-Rate* — also die Anzahl im Cache befindlicher Dateien — im Vergleich zu heute üblichen Cache-Ersetzungsstrategien signifikant gesteigert werden kann. Auch diese Steigerung wird durch Nutzung skalierbarer Medien erreicht, indem nicht direkt die komplette Datei aus dem Cache entfernt wird, wenn dieser seinen maximalen Füllstand erreicht hat und weitere Daten aufgenommen werden sollen. Vielmehr werden nur Teile der Datei entfernt, bspw. die höchste Qualitätsschicht. Analog zu den bereits vorgestellten Verfahren werden die fehlenden Daten bei einem erneuten Aufruf durch die Substitutionsmethode kompensiert, so dass ein Aufruf der Datei für die nachfolgenden Prozesse transparent ist.

Ein Vergleich mit dem Stand der Technik, eine kritische Diskussion und Beurteilung der Ergebnisse sowie ein Ausblick für zukünftige wissenschaftliche Arbeiten auf dem adressierten Forschungsgebiet schließen die vorliegende Arbeit ab.

8. Schlussbetrachtung

Skalierbare Codecs existieren schon seit mehreren Jahrzehnten. Nahezu jeder Anwender, der mit skalierbaren Codecs arbeitet, erkennt die Vorteile und Möglichkeiten, die diese Technologie bietet. Jedoch konnten sich skalierbare Codecs bisher nur in Randbereichen der Multimedia-Welt etablieren. Vorrangig werden sie bislang im Web-Umfeld eingesetzt. Mit Durchführung und Veröffentlichung dieser Arbeit möchte der Autor zeigen, dass die Skalierbarkeit auch innerhalb von Computersystemen genutzt werden kann, um bestimmte Arbeitsabläufe im Bereich der professionellen und semiprofessionellen Filmproduktion zu verbessern. Diese Verbesserungen können auf der einen Seite z. B. die Echtzeitfähigkeit von bestimmten Prozessen sein, die ohne skalierbare Medien nicht echtzeitfähig wären. Auf der anderen Seite kann die Skalierbarkeit genutzt werden, um vorhandene Verfahren für die Benutzer zu vereinfachen. Diese Arbeit zeigt dabei neue Möglichkeiten und Verfahren, die erst durch die Nutzung skalierbarer Codecs möglich sind. Hierdurch kann sie dazu beitragen, die Grundtechnologie der Skalierbarkeit weiter zu bewerben. Ob diese Arbeit den langersehnten Durchbruch für dieses Themengebiet liefern kann, hängt sicherlich auch von der Frage ab, ob die vorgestellten Technologien zeitnah in marktreife Applikationen integriert werden können.

Abkürzungen

AAC	Advanced Audio Coding
ACL	Access Control List
ADS	Alternate Data Stream
AGP	Accelerated Graphics Port
API	Application Programming Interface
ARAID	Adaptive RAID
ATA	Advanced Technology Attachement, Adavanced Technology Attachement
AVC	Advanced Video Coding
AVI	Audio Video Interleave
BSAC	Bitsliced Arithmetic Coding
BSD	Berkeley Software Distribution
CD	Compact Disc
CfP	Call for Papers
CGS	Coarse Grain Scalability
CPRL	Component-Position-Resolution-Layer Progressionsreihenfolge
CPU	Central Processing Uni
DCP	Digital Cinema Package
DP	Display Port
DSA	DateiSystem-Anwendung
DVD	Digital Versatile Disc
DVI	Digital Visual Interface
DWT	Discrete Wavelet Transformation
EBCOT	Embedded Block Coding with Optimal Truncation
EEPROM	Electrically Erasable Programmable Read Only-Memory
EOC	End Of Codestream
EOF	End Of File
FAT	File Allocation Table
FGS	Fine Grain Scalability
FIFO	First In First Out
FSA	File System Application
FTP	File Transfer Protocol
FPS	Frames Per Second
FUSE	Filesystem in UserspacE
GIF	Graphics Interchange Format
GPU	Graphics Processing Unit
HD	Hard Disk
HDD	Hard Disk Drive, Hard Disk Drive
HEVC	High Efficiency Video Coding
HTTP	HyperText Transfer Protocol
HVS	Human Visual System
IAP	Intermediate Archive Package

ICT	Irreversible Color Transformation
IDE	Integrated Device Electronic
IPTV	Internet Protocol Television
ISO	International Standardization Organization
ITU	International Telecommunication Union
JPEG	Joint Photographic Experts Group
JPIP	JPEG 2000 Interactive Protocol
JVT	Joint Venture Team
LCN	Logical Cluster Number
LFU	Least Frequently Used
LRCP	Layer-Resolution-Component-Position Progressionsreihenfolge
LTO	Long Term Open
MAP	Master Archive Package
MFT	Master File Table
MGS	Medium Grain Scalability
MPEG	Moving Picture Experts Group
MPEG-DASH	Dynamic Adaptive Streaming over HTTP
MRK	Media Repackaging Komponente
MVC	Multi View Coding
MXF	Material eXchange Format
NAND	Not AND
NAS	Network Attached Storage
NOR	Not OR
NTFS	New Technology File System
PCI	Peripheral Component Interconnect
PCI-E	Peripheral Component Interconnect Express
PCI-X	PCI eXtended
PCRL	Position-Component-Resolution-Layer Progressionsreihenfolge
PDA	Personal Digital Assistant
PLM	Packet Length Marker
PLT	Packet Length Tile Marker
PSNR	Peak Signal-to-Noise Ratio
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory
RCT	Reversible Color Transformation
RLCP	Resolution-Layer-Component-Position Progressionsreihenfolge
ROI	Region Of Interest
ROM	Read Only Memory
RPCL	Resolution-Position-Component-Layer Progressionsreihenfolge
RPM	Rounds Per Minute
RVC	Reconfigurable Video Coding
SATA	Serial Advanced Technology Attachment
SCSI	Small Computer System Interface
SFS	Scalable File System
SLS	Scalable To Lossless
SSD	Solid State Drive

STTL	Scalable Time To Live
SVC	Scalable Video Coding, Scalable Video Coding
TCP/IP	Transmission Control Protocol/Internet Protocol
UHD	Ultra High Definition
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Institute of Electrical and Electronics Engineers, Universal Serial Bus
VCN	Virtual Cluster Number
VFS	Virtual File System
VGA	Video Graphics Array
WiMAX	Worldwide Interoperability for Microwave Access

Literaturverzeichnis

[1] T. Acharya and P. Tsai, *JPEG2000 : Standard for Image Compression*. Chichester: Wiley, 2005.

[2] Advantech. (2013, June) DSPC-8681. [Online]. http://www.advantech.com/products/DSPC-8681/mod_A9314996-0022-4927-9AA4-6B1060D4E5E8.aspx, Zugriffsdatum: 07.06.2013.

[3] K. Brandenburg and B. Grill, "First Ideas on Scalable Audio Coding," in *97th Convention of the AES*, San Francisco, 1994, S. 10-13.

[4] R. Brause, *Betriebssysteme – Grundlagen und Konzepte*. Berlin Heidelberg: Springer, 2001.

[5] V. Bruns, H. Sparenberg, and S. Fößel, "Videocodierer und Verfahren zum Decodieren einer Sequenz von Bildern," DE-Patent DE 10 2010 030 973 A1, Feb. 02, 2012.

[6] D. Bryant. (2013, June) WavPack. [Online]. http://www.wavpack.com/, Zugriffsdatum: 10.06.2013.

[7] CCITT, "Digital Compression and Coding of Continuous-tone Still Images – Requirements and Guidelines," 1992.

[8] K. Chandra and A.R. Reibman, "Modeling one- and two-layer variable bit rate video," in *IEEE/ACM Transactions on Networking,* vol. 7, 3, S. 398-413.

[9] E. Chang and A. Zahkor, "Scalable video coding using 3-D subband velocity coding and multirate quantization," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 5, Minneapolis, 1993, S. 574-577.

[10] M.Y.M. Chiu and K.-H.A. Yeung, "Partial video sequence caching scheme for VOD systems with heterogeneous clients," in *IEEE Transactions on Industrial Electronics*, vol. 45, 1, S. 44-51.

[11] Thunderbolt Community. (2012, Aug.) Thunderbolt TM Technology. [Online]. https://thunderbolttechnology.net/, Zugriffsdatum: 03.06.2014.

[12] A. Dan and D. Sitaram, "Multimedia Caching Strategies for Heterogeneous Application and Server Enviroments," *Multimedia Tools and Applications*, vol. 4, no. 3, S. 279-312,

May 1997.

[13] A. Descampe, C. De Vleeschouwer, M. Iregui, B. Macq, and F. Marqués, "Prefetching and Caching Strategies for Remote and Interactive Browsing of JPEG2000 Images," in *IEEE Transactions on Image Processing*, vol. 16, 2007, S. 1339-1354.

[14] A. Descampe, C. De Vleeschouwer, M. Iregui, B. Macq, and F. Marques, "Prefetching Strategies for Remote and Interactive Browsing of JPEG2000 Images," in *International Conference on Image Processing, 2006. ICIP*, 2006, S. 3201-3204.

[15] A. Descampe, J. Ou, P. Chevalier, and B. Macq, "Data Prefetching for Smooth Navigation of Large Scale JPEG 2000 Images," in *IEEE International Conference on Multimedia and Expo (ICME)*, 2005.

[16] S. Deshpande and W. Zeng, "HTTP Streaming of JPEG2000 Images," in *Information Technology: Coding and Computing,* 2001, S. 15-19.

[17] S. Deshpande and W. Zeng, "Scalable Streaming of JPEG2000 Images using Hypertext Transfer Protocol," in *In Proc. ACM Multimedia*, 2011.

[18] Digital Cinema Initiatives (DCI). (2012, August) Digital Cinema System Specification Version 1.2 with Errata as of 30 August 2012 Incorporated. [Online]. http://dcimovies.com/specification/DCI_DCSS_v12_with_errata_2012-1010.pdf, Zugriffsdatum: 24.06.2014.

[19] J.-B. Ernst-Desmulier, D. Charlet, P. Chatonnay, and F. Spies, "A Peer-to-Peer Approach for Cache Sibling," in *Proc. on Distributed Frameworks for Multimedia Applications*, 2005, S. 323-330.

[20] R. Geiger, *Audio Coding Based on Integer Transforms*., 2004.

[21] F. Ghido. (2011, Feb.) Ghido's Data Compression Page. [Online]. http://www.losslessaudio.org/, Zugriffsdatum: 10.06.2013.

[22] D. Giampaolo, *Practical File System Design*. San Francisco: Morgan Kaufmann Publishers, 1999.

[23] S. Gilbertson. (2013, Jan.) The Return of the Progressive JPEG. [Online]. http://www.webmonkey.com/2013/01/the-return-of-the-progressive-jpeg/, Zugriffsdatum: 06.06.2013.

[24] B. Grill, "A Bit Rate Scalable Perceptual Coder for MPEG-4 Audio," in *103rd*

Convention of the AES, New York, 1997.

[25] B. Grill, "The MPEG-4 General Audio Coder," in *Proceedings of the AES 17th International Conference*, Florence, Italy, 1999, S. 147-156.

[26] B. Grill and K. Brandenburg, "A Two- or Three-Stage Bit Rate Scalable Audio Coding System," in *99th Convention of the AES*, New York, 1995.

[27] Network Working Group. (1999, June) Hypertext Transfer Protocol -- HTTP/1.1. [Online]. http://www.ietf.org/rfc/rfc2616.txt, Zugriffsdatum: 03.06.2014.

[28] The Moving Picture Experts Group. (2012, July) Joint Call for Proposals on Scalable Video Coding Extensions of High Efficiency Video Coding (HEVC). [Online]. http://mpeg.chiariglione.org/sites/default/files/files/standards/parts/docs/w12957-v2-w12957.zip, Zugriffsdatum: 21.07.2013.

[29] H. Guo and K.-T. Lo, "Cooperative Media Data Streaming with Scalable Video Coding," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, 2008, S. 1273-1281.

[30] F. Hartanto, J. Kangasharju, M. Reisslein, and K. W. Ross, "Caching video objects: layers vs versions?," in *2002 IEEE International Conference on Multimedia and Expo, 2002. ICME '02.*, vol. 2, 2002, S. 45-48.

[31] John L. Hennessy and David A. Patterson, *Computer Architecture – A Quantitave Approach – Fourth Edition*. San Francisco: Morgan Kaufmann Publishers, 2007.

[32] Hewlett-Packard, IBM and Quantum. (2012, November) LTO Ultrium Generations. [Online]. http://www.lto.org/technology/generations.html, Zugriffsdatum: 14.12.2012.

[33] Hiroki. (2011, February) Dokan Project 2007. [Online]. http://dokan-dev.net/en, Zugriffsdatum: 30.05.2014.

[34] D. Hong, D. De Vleeschauwer, and F. Baccelli, "A chunk-based caching algorithm for streaming video," in *Proceedings of the 4th Workshop on Network Control and Optimization*, Ghent, Belgium.

[35] K.-M. Ho, W.-F. Poon, and K.T. Lo, "Performance Study of Large-Scale Video Streaming Services in Highly Heterogeneous Environment," *IEEE Transactions on Broadcasting*, vol. 53, no. 4, S. 763-773, Nov. 2007.

[36] Avaya Inc. Scalable Video Coding (SVC Video). [Online]. http://www.radvision.com/Solutions/Video-Communications-Technology/Scalable-

Video-Coding/default.htm, Zugriffsdatum: 17.06.2013.

[37] Information and Communication Theory Group (ICT) of Delft University of Technology. (2011, Jan.) Image and Video Compression Learning Tool VcDemo. [Online]. http://msp.ewi.tudelft.nl/content/image-and-video-compression-learning-tool-vcdemo, Zugriffsdatum: 14.05.2014.

[38] International Standard 14496-3:2001, "Information technology - Coding of audio-visual objects - Part 3: Audio," 2001.

[39] ISO/IEC 14492-2 (MPEG-4 Visual), "Coding of audio-visual objects-part2: Visual," Version 1: Apr. 1999, Version 2: Feb. 2000, Version 3: May 2004.

[40] ISO/IEC 14496-12. (2008) Coding of audio-visual objects — Part 12: ISO base media file format.

[41] ISO/IEC 15444-1. (2000) Information technology – JPEG 2000 image coding system – Part 1: Core coding system.

[42] ISO/IEC 15444-1:2004/Amd 2:2009. (2009) Extended profiles for cinema and video production and archival applications.

[43] ISO/IEC 15444-9:2005. (2005) Information technology — JPEG 2000 image coding system: Interactivity tools, APIs and protocols.

[44] ISO/IEC JTC 1/SC 29/WG 11 (MPEG), "Dynamic adaptive streaming over HTTP," vol. 23001-6, Guangzhou, China, Oct. 2012.

[45] ISO/IEC JTC1/SC29/WG1 N505, *Call for contributions for JPEG2000 (JTC1.29.14,15444): image coding system*., March 1997.

[46] ITU-T Rec. H.261, "Video Codec for Audiovisual Services at p × 64 kbit/s," Mar. 93.

[47] ITU-T Rec. H.262 and ISO/IEC 13818-2 (MPEG-2 Video), "Information technology – Generic coding of moving pictures and associated audio information: Video," 1995.

[48] ITU-T Rec. H.263, "Video Coding for Low Bit Rate Communication," Version 1: Nov. 1995, Version 2: Jan. 1998, Version 3: Nov. 2000.

[49] ITU-T Rec. H.264, "Advanced video coding for generic audiovisual services," 2005, S. 250-251.

[50] M. Jelinek, T. Vaillancourt, and J. Gibbs, "G.718: A new embedded speech and audio

coding standard with high resilience to error-prone transmission channels," *IEEE Communications Magazine*, vol. 47, no. 10, S. 117-123, Oct 2009.

[51] J. Kangasharju, F. Hartanto, M. Reisslein, and K.W. Ross, "Distributing layered encoded video through caches," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, Anchorage, AK, 2001, S. 1791-1800.

[52] J. Kangasharju, Y. G. Kwon, and A. Ortega, "Design and implementation of a soft caching strategy," *Computer Networks and ISDN Systems*, vol. 30, no. 22-23, S. 2113-2121, November 1998.

[53] S. Kang, W. Youjip, and S. Roh, "Harmonic Placement: File System Support for Scalable Streaming of Layer Encoded Object," *NOSSDAV 2006*, 2006.

[54] J. Lee, T. Kim, and S. Ko, "Motion Prediction Based on Temporal Layering for Layered Video Coding," in *Proc. ITC-CSCC*, 1998, S. 245-248.

[55] W. Li, "Overview of Fine Granularity Scalability in MPEG-4 Video Standard," in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, 2001, S. 301-317.

[56] J. Li, "Vfile presentation at JPEG 2000 Maui meeting," Maui, 1999.

[57] S.T. Liang and T.-S. Chang, "A bandwidth effective streaming of JPEG2000 images using hypertext transfer protocol," in *Multimedia and Expo, 2002. ICME '02. Proceedings*, 2002, S. 525-528.

[58] L. Lima, D. Taubman, and R. Leonardi, "JPIP Proxy Server for remote browsing of JPEG2000 images," in *10th Workshop on Multimedia Signal Processing*, Cairns, Qld, 2008, S. 844-849.

[59] C.-L. Lin, H.-H. Lee, C.-L. Chan, and J.-S. Wang, "Cooperative Proxy Framework for Layered Video Streaming," in *Global Telecommunications Conference*, vol. 1, 2005, S. 251-255.

[60] J. Li and H.-H. Sun, "On interactive browsing of large images," in *IEEE Transactions on Multimedia*, vol. 5, No. 4, Redmond, 2003, S. 581-590.

[61] J. Liu, X. Chu, and J. Xu, "Proxy Cache Management for Fine-Grained Scalable Video Streaming," in *Proc. on INFOCOM 2004*, vol. 3, Hong Kong, S. 1490-1500.

[62] J. Liu, J. Xu, and X. Chu, "Fine-Grained Scalable Video Caching for Heterogeneous

Clients," *IEEE Transactions on Multimedia*, vol. 8, no. 5, S. 1011-1020, Oct. 2006.

[63] B. Liu, W. Zhang, and S. Yu, "Proxy Caching Based on Segments for Layered Encoded Video," in *Proceedings of the IEEE 6th Circuits and Systems Symposium on Emerging Technologies: Frontiers of Mobile and Wireless Communication*, vol. 1, 2004, S. 41-44.

[64] Klaus Manhart. (2014, Mar.) Alle Wege führen nach unten. [Online]. http://www.computerwoche.de/a/alle-wege-fuehren-nach-unten,2556478, Zugriffsdatum: 12.06.2014.

[65] A. Mathur et al., "The new ext4 filesystem: current status and future plans," in *Ottowa Linux Symbosium, vol.2*, 2007, S. 21-34.

[66] S. McCanne, "Scalable compression and transmission of internet multicast video," *Ph.D. dissertation*, Dec. 1996.

[67] S. McCanne and M. Vetterli, "Joint source/channel coding for multicast packet video," in *International Conference on Image Processing, 1995. Proceedings.*, vol. 1, Washington, DC, 1995, S. 25-28.

[68] S. Miklos and H. Csaba. (2011, February) Filesystem in Userspace. [Online]. http://sourceforge.net/projects/fuse/, Zugriffsdatum: 03.06.2014.

[69] J.L. Monteagudo-Pereira, F. Auli-Llinas, J. Serra-Sagrista, and J. Bartrina-Rapesta, "Smart JPIP Proxy Server with Prefetching Strategies," in *Data Compression Conference*, Snowbird, UT, 2010, S. 99-108.

[70] J.L. Monteagudo-Pereira et al., "Enhanced transmission of JPEG2000 imagery through JPIP proxy and user-navigation model," in *Data Compression Conference (DCC)*, Snowbird, UT, S. 22-31.

[71] C. Müller and S.,Timmerer,C. Lederer, "An evaluation of dynamic adaptive streaming over HTTP in vehicular environments," in *Proc. of the 4th Workshop on Mobile Video (MoVid)*, Proceedings of the 4th Workshop on Mobile Video, S. 37-42.

[72] C. Müller, D. Renzi, S. Lederer, S. Battista, and C. Timmerer, "Using Scalable Video Coding for Dynamic Adaptive Streaming Over HTTP in Mobile Environments," in *Proc. on 20th European Signal Processing Conference*, Bucharest, Romania, 2012, S. 2208-2212.

[73] NEMA. (2014, May) DICOM - Digital Imaging and Communications in Medicine. [Online]. http://medical.nema.org/, Zugriffsdatum: 06.06.2013.

[74] Nevion. (2013, June) Nevion. [Online]. http://www.nevion.com/products/tvg/tvg450, Zugriffsdatum: 07.06.2013.

[75] J.-R. Ohm, "Advances in Scalable Video Coding," *Proceedings of the IEEE*, vol. 93, no. 1, S. 42-56, Jun 2005.

[76] The Serial ATA International Organization. (2007, Feb.) Serial ATA Revision 2.6. [Online]. http://www.sata-io.org/, Zugriffsdatum: 29.05.2014.

[77] J.P. Ortiz, V.G. Ruiz, and I. Garcia, "An Efficient Technique for Remote Browsing of JPEG 2000 Images on the Web," in *Computación de altas prestaciones: actas de las XV Jornadas de Paralelismo*, Almeria, 2004, S. 322-327.

[78] J.P. Ortiz, V.G. Ruiz, M.F. Lopez, and I. Garcia, "Interactive Transmission of JPEG2000 Images Using Web Proxy Caching," in *IEEE Transactions on Multimedia*, vol. 10, 2008, S. 629-636.

[79] S. Park, Y. Kim, S Kim, and Y. Seo, "Multi-Layer Bit-Sliced Bit- Rate Scalable Audio Coding," in *103rd Convention of the AES*, New York, 1997.

[80] D.A. Patterson, G. Gibson, and R.H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," in *SIGMOD '88 Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, New York, NY, USA, 1988, S. 109-116.

[81] K. Paulsen, *Moving Media Storage Technologies*. Burlington: Elsevier, 2011.

[82] W.B. Pennebaker and J.L. Mitchell, *JPEG: Still Image Data Compression Standard*.: Springer, 1993.

[83] S. Podlipnig and L. Boszormenyi, "Replacement strategies for quality based video caching," in *2002 IEEE International Conference on Multimedia and Expo, 2002. ICME '02*, vol. 2, 2002, S. 49-52.

[84] E.A. Politou, G.P. Pavlidis, and C. Chamzas, "JPEG2000 and Dissemination of Cultural Heritage Over the Internet," in *IEEE Transactions on Image Processing*, vol. 13, 2004, S. 293-301.

[85] A. Rajgarhia and A Gehani, "Performance and Extension of User Space File Systems," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, New York, S. 206-213.

[86] R. Rejaie, M. Handley, and D. Estrin, "Layered Quality Adaptation for Internet Video Streaming," in *IEEE Journal on Selected Areas in Communications*, vol. 18, 12, S. 2000.

[87] R. Rejaie, H. Handley, and D. Estrin, "Quality Adaptation for Congestion Controlled Video Playback over the Internet," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, vol. 29, 4, S. 189-200.

[88] R. Rejaie and J. Kangasharju, "Mocha: a quality adaptive multimedia proxy cache for internet streaming," in *NOSSDAV '01 Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, S. 3-10.

[89] R. Rejaie, H. Yu, M. Handley, and D. Estrin, "Multimedia Proxy Caching Mechanism for Quality Adaptive Streaming Applications in the Internet," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, Tel Aviv, S. 980-989.

[90] I.E. Richardson, *The H.264 Advanced Video Compression Standard – Second Edition*. Chichester: Wiley, 2010.

[91] J. Rieck, "Scalable Video for Peer-to-Peer Streaming," Technical University of Vienna, Vienna, Master-Thesis 2008.

[92] R. Russon and Y. Fledel. (2005) NTFS Documentation. [Online]. http://ftp.kolibrios.org/users/Asper/docs/NTFS/ntfsdoc.html, Zugriffsdatum: 03.06.2014.

[93] intoPIX, s.a. (2013, June) http://www.intopix.com. [Online]. http://www.intopix.com/products/index/index/id/17/lang/en#.UbHSTxm2w4S, Zugriffsdatum: 07.06.2013.

[94] Y. Sánchez et al., "iDASH: Improved Dynamic Adaptive Streaming over HTTP using Scalable Video Coding," in *Proceedings of the second annual ACM conference on Multimedia systems MMSys '11*, S. 257-264.

[95] Y. Sánchez et al., "Improved caching for HTTP-based Video on Demand using Scalable Video Coding," in *Proc. on Consumer Communications and Networking Conference*, Las Vegas, NV, 2011, S. 595-599.

[96] P. Schelkens, S. Athanassios, and E. Touradj, *The JPEG 2000 Suite*. Chichester: Wiley, 2009.

[97] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9, S. 1103-1120, Sept. 2007.

[98] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the H.264/AVC standard," in *IEEE Transactions on Circuits and Systems for Video Technology, vol. 17, no. 9*, Sep. 2007, S. 1103-1120.

[99] S. Shunya and K. Mei, "Application of Video Scalability to Video Data Distribution Network and its Evaluation," in *2004 IEEE International Symposium on Industrial Electronics*, S. 305-109.

[100] Irfan Skiljan. (2013) IrfanView - Official Homepage - one of the most popular viewers worldwide. [Online]. www.irfanview.com, Zugriffsdatum: 20.12.2013.

[101] PassMark Software. (2013) PassMark PerformanceTest - PC benchmark software. [Online]. http://www.passmark.com/products/pt.htm, Zugriffsdatum: 20.12.2013.

[102] SourceForge. (2012, November) File systems using FUSE. [Online]. http://sourceforge.net/apps/mediawiki/fuse/index.php?title=FileSystems, Zugriffsdatum: 03.06.2014.

[103] H. Sparenberg, V. Bruns, and S. Foessel, "Use-Case-Optimized Data Storage of Scalable Media Files," in *Third International Conference on Innovative Computing Technology (INTECH)*, London, 2013, S. 35-39.

[104] H. Sparenberg and S. Foessel, "Advanced Storage Techniques using Scalable Media," in *SMPTE Conference Proceedings*, Hollywood, USA, 2013, S. 1-6.

[105] H. Sparenberg and S. Foessel, "Real Time File System for Content Distribution," in *SMPTE Conference Proceedings*, Hollywood, USA, 2012, S. 1-7.

[106] H. Sparenberg, T. Joormann, C. Feldheim, and S. Foessel, "Adaptive Raid: Introduction of Optimized Storage Techniques for Scalable Media," in *20th IEEE International Conference on Image Processing (ICIP)*, Melbourne, 2013, S. 1826-1830.

[107] H. Sparenberg, M. Martin, and S. Foessel, "Introduction of Eviction Strategies for Caching Scalable Media Files," in *Digital Information Management (ICDIM), 2012 Seventh International Conference on*, Macau, 2012, S. 352-356.

[108] H. Sparenberg, M. Martin, and S. Foessel, "Real-time Capable File System for Scalable Media," in *Picture Coding Symposium (PCS)*, Krakau, 2012, S. 345-348.

[109] H. Sparenberg, A. Schmitt, R. Scheler, S. Foessel, and K. Brandenburg, "Virtual file system for scalable media formats: Architecture proposal for managing and handling scalable media files," in *14th ITG Conference on Electronic Media Technology (CEMT)*, Dortmund, 2011, S. 1-5.

[110] S. Stefanov. (2008, Dec.) Image Optimization, Part 4: Progressive JPEG…Hot or Not? [Online]. http://www.yuiblog.com/blog/2008/12/05/imageopt-4/, Zugriffsdatum: 06.06.2013.

[111] Z. Su, Z. Zhang, and Y. Chen, "Retrieve Scalable Image Contents over Image Contents Network by Efficient Caching Algorithms," in *3rd International Conference on Innovative Computing Information and Control, 2008. ICICIC '08.*, vol. 1, Dalian, Liaoning, 1, S. 362-365.

[112] D. Taubman, "Remote browsing of JPEG2000 images," in *2002 International Conference on Image Processing. Proceedings*, Sydney, 2002, S. 229-232.

[113] D. Taubman and M. W. Marcellin, *JPEG2000 : Image Compression Fundamentals, Standards and Practice*. Boston: Kluwer Academic Publishers, 2002.

[114] D. Taubman and R. Prandolini, "Architecture, philosophy, and performance of JPIP: internet protocol standard for JPEG2000," in *SPIE Proceedings Vol. 5150*, Switzerland, 2003, S. 791-805.

[115] D. Taubman and R. Rosenbaum, "Rate-distortion optimized interactive browsing of JPEG2000 images," in *International Conference on Image Processing, 2003. ICIP 2003. Proceedings*, 2003.

[116] Vidyo Personal Telepresence. Technology Platform - Internet Video Conferencing. [Online]. http://www.vidyo.com/technology/, Zugriffsdatum: 17.06.1013.

[117] TG Publishing GmbH. (2012, November) AS-SSD 4K Q64 Random Read. [Online]. http://www.tomshardware.de/charts/ssd-charts-2012/AS-SSD-4K-Q64-Random-Read,2786.html, Zugriffsdatum: 13.12.2012.

[118] TG Publishing GmbH. (2012, November) Read Throughput Maximum. [Online]. http://www.tomshardware.de/charts/hdd-charts-2012/-02-Read-Throughput-Maximum-h2benchw-3.16,2900.html, Zugriffsdatum: 13.12.2012.

[119] I. Unanue et al., *A Tutorial on H.264/SVC Scalable Video Coding and its Tradeoff between Quality, Coding Efficiency and Performance.*, 2011.

[120] USB Implementers Forum, Inc. (2000, Apr.) Universal Serial Bus Specification – Revision 2.0. [Online]. http://www.usb.org, Zugriffsdatum: 31.01.2014.

[121] USB Implementers Forum, Inc. (2011, Apr.) Universal Serial Bus Specification – Revision 3.0. [Online]. http://www.usb.org, Zugriffsdatum: 01.02.2014.

[122] USB Implementers Forum, Inc. (1998, September) Universal Serial Bus Specification – Revision 1.1. [Online]. http://www.usb.org, Zugriffsdatum: 31.01.2014.

[123] C. Valens. (2010, Oct.) A Really Friendly Guide to Wavelets. [Online]. http://polyvalens.pagesperso-orange.fr/clemens/wavelets/wavelets.html, Zugriffsdatum: 14.05.2014.

[124] M. Vishwanath and P. Chou, "An efficient algorithm for hierarchical compression of video," in *IEEE International Conference on Image Processing, 1994. Proceedings.*, Austin, TX , 1994, S. 275-279.

[125] G.K. Wallace, "The JPEG still picture compression standard ," *IEEE Transcations of Consumer Electronics*, vol. 38, no. 1, S. 18-34, Feb. 1992.

[126] J.Z. Wang and P.S. Yu, "Fragmental Proxy Caching for Streaming Multimedia Objects," *IEEE Transactions on Multimedia*, vol. 9, no. 1, S. 147-156, Dec. 2006.

[127] C. Wiley, "DisplayPort Technical Overview," in *IEEE International Conference on Consumer Electronics (ICCE)*, Jan. 2011.

[128] Y. Won, S. Yang, and S. Kang, "Harmonic data placement: file system support for scalable streaming," *12th International Conference on Multi-Media Modelling Conference Proceedings*, S. 481-484, 2006.

[129] P.-H. Wu and Y.H. Hu, "Optimal Layered Video IPTV Multicast Streaming Over Mobile WiMAX Systems," in *IEEE Transactions on Multimedia*, vol. 13, 6, S. 1395-1403.

[130] K.-L. Wu, P.S. Yu, and J.L. Wolf, "Segmentation of Multimedia Streams for Proxy Caching," *IEEE Transactions on Multimedia*, vol. 6, no. 5, S. 770-780, 2004.

[131] R. Yu et al., "MPEG-4 Scalable to Lossless Audio Coding," in *117th Convention of the AES*, San Francisco, 2004.

[132] R. Yu, X. Lin, S. Rahardja, and C.C. Ko, "A Scal- able to Lossless Audio Coder for MPEG-4 Lossless Audio Coding," in *Proc. of the ICASSP*, Montreal, Canada, 2004.

[133] M. Zink, O. Heckmann, J. Schmitt, and A. Mauthe, "Polishing: a technique to reduce variations in cached layer-encoded video," in *29th Proc. on Euromicro Conference*, vol. 1, 2003, S. 249-254.

[134] M. Zink, O. Künzel, J. Schmitt, and R. Steinmetz, "Subjective Impression of Variations in Layer Encoded Videos," in *Proc. IWQoS '03*, Monterey, CA, 2003, S. 137-154.

[135] M. Zink, J. Schmitt, and C. Griwodz, "Layer-encoded video streaming a proxy's perspective," *Communications Magazine, IEEE*, vol. 42, no. 8, S. 96-103, Aug. 2004.

[136] M. Zink, J. Schmitt, and R. Steinmetz, "Layer-encoded video in scalable adaptive streaming," *IEEE Transactions on Multimedia*, vol. 7, S. 75-84, Jan. 2005.

[137] M. Zink, J. Schmitt, and R. Steinmetz, "Retransmission scheduling in layered video caches," in *IEEE International Conference on Communications, 2002*, vol. 4, 2002, S. 2474-2478.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang

A. Funktionsweise der Prozedur „Get Scalable Media Plugin“

Die Klasse *UserManagement* (vgl. Abbildung 56) kapselt dabei die Aufrufe zur Datenbank und liefert im Erfolgsfall den Benutzer und dessen Zugriffsrechte. Handelt es sich bei der angefragten Datei um eine skalierbare Medienvariante, die vom System interpretiert werden kann, werden die Parameter *User* sowie der reale Dateiname verwendet, um ein entsprechendes Plug-In zu erstellen. Dieses Plug-In dient später zur Generierung der Dateivarianten, in Abhängigkeit der Benutzerrechte von *User*. Im Erfolgsfall kann mit einem Zeiger auf ein Objekt vom Typ *JPEG2000ScalableMediaPlugin* nachfolgend eine skalierte Datei erstellt werden.

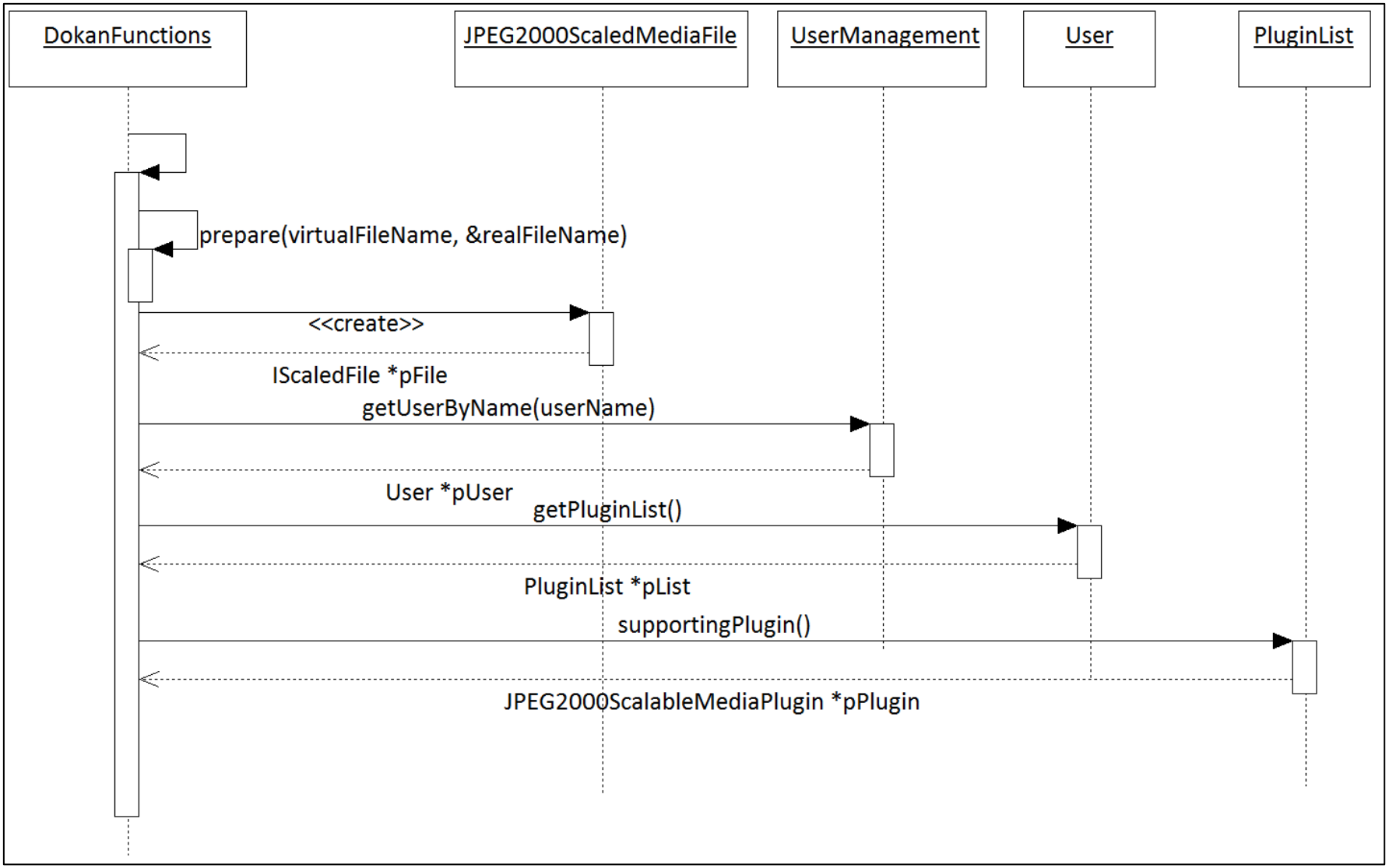

Abbildung 56: Sequenzdiagramm "Get Scalable Media Plugin" im virtuellen Dateisystem für skalierbare Medien

B. Funktionsweise der Prozedur „Generate Scaled Media File“

Durch den Aufruf der Funktion *createScaledMediaFile* der Klasse *JPEG2000ScalableMediaPlugin* (vgl. Abbildung 57) erzeugt diese zunächst ein neues Objekt der Klasse *JPEG2000ScalableMediaFile*, sowie ein Objekt der Klasse *JPEG2000PictureImageOptions*. Beide Objekte werden mit Standardwerten initialisiert, so dass die neue Variante zunächst die Eigenschaften der originalen JPEG 2000-Datei erbt. Die Instanz der Klasse *JPEG2000ScalableMediaPlugin* beinhaltet dabei die Zugriffsrechte des angemeldeten Benutzers und sorgt mittels Aufruf der Funktion *applyPermissions* für die Einhaltung entsprechender Vorgaben. Hierbei wird das Bild bei Bedarf in Auflösung und/oder Qualität reduziert. Anschließend veranlasst das Objekt *JPEG2000ScalableMediaFile* durch den Aufruf der Funktion *updateMetadata* eine Aktualisierung der beschreibenden Daten. Dabei kommt eine Cache-Funktion zum Einsatz, die prüft, ob die aktuelle Variante der Daten in der Vergangenheit schon einmal angefragt wurde. Hierzu ruft das Objekt der skalierbaren Dateivariante die Funktion *getVariant* des Objektes *JPEG2000MetadataCache* auf. Durch die Parameter des Dateipfades zur originalen Datei kann der Cache ermitteln, ob bereits eine solche Variante im Vorfeld berechnet wurde und das Ergebnis aus dem Cache bereitgestellt werden kann. Im Erfolgsfall wird ein Objekt vom Typ *JPEG2000ScalableMediaFile* zurückgegeben. Andernfalls muss die skalierbare Dateivariante neu erstellt werden. Das Ergebnis der Berechnung wird anschließend im Cache gespeichert, um nachfolgende Anfragen schneller bedienen zu können. Über das Plug-In wird die skalierbare Dateivariante zurück an die aufrufende Funktion gereicht.

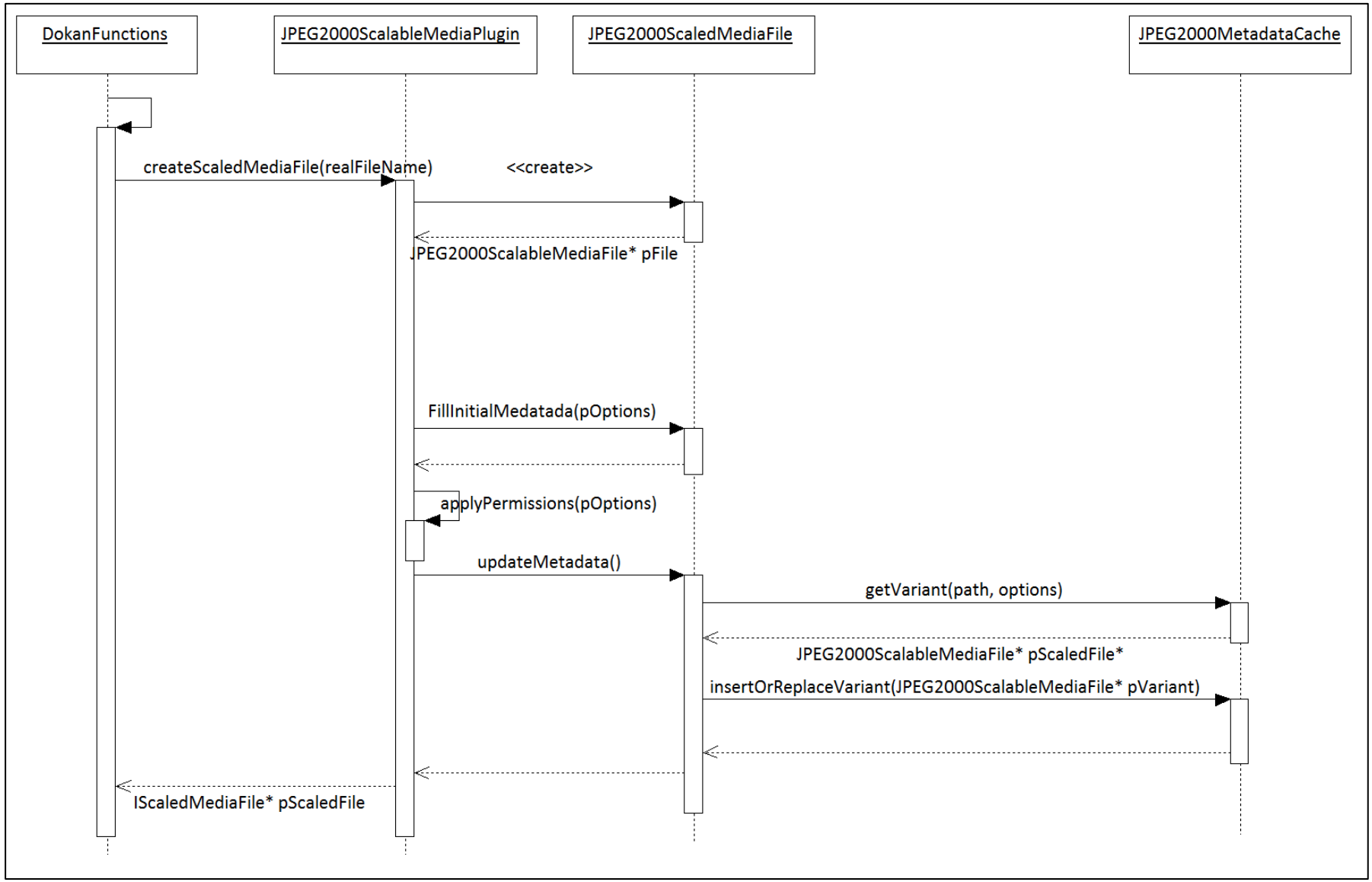

Abbildung 57: Sequenzdiagramm "Generate Scaled Media File" im virtuellen Dateisystem für skalierbare Medien

C. Funktionsweise der Prozedur „Read File“

Analog zur Ausführung der Funktion *GetFileInformation* (vgl. Abbildung 22) wird in Abbildung 58 eine Instanz der Klasse *JPEG2000ScaledMediaFile* erstellt, deren Daten die Dateivariante für die aktuellen Zugriffsrechte wiedergeben. Im Gegensatz zur bereits geschilderten Variante wird an dieser Stelle jedoch die Funktion *readData* aufgerufen, wodurch letztendlich Daten aus der Instanz *JPEG2000PictureVariant* gelesen werden und an den aufrufenden Prozess zurückgeliefert werden können.

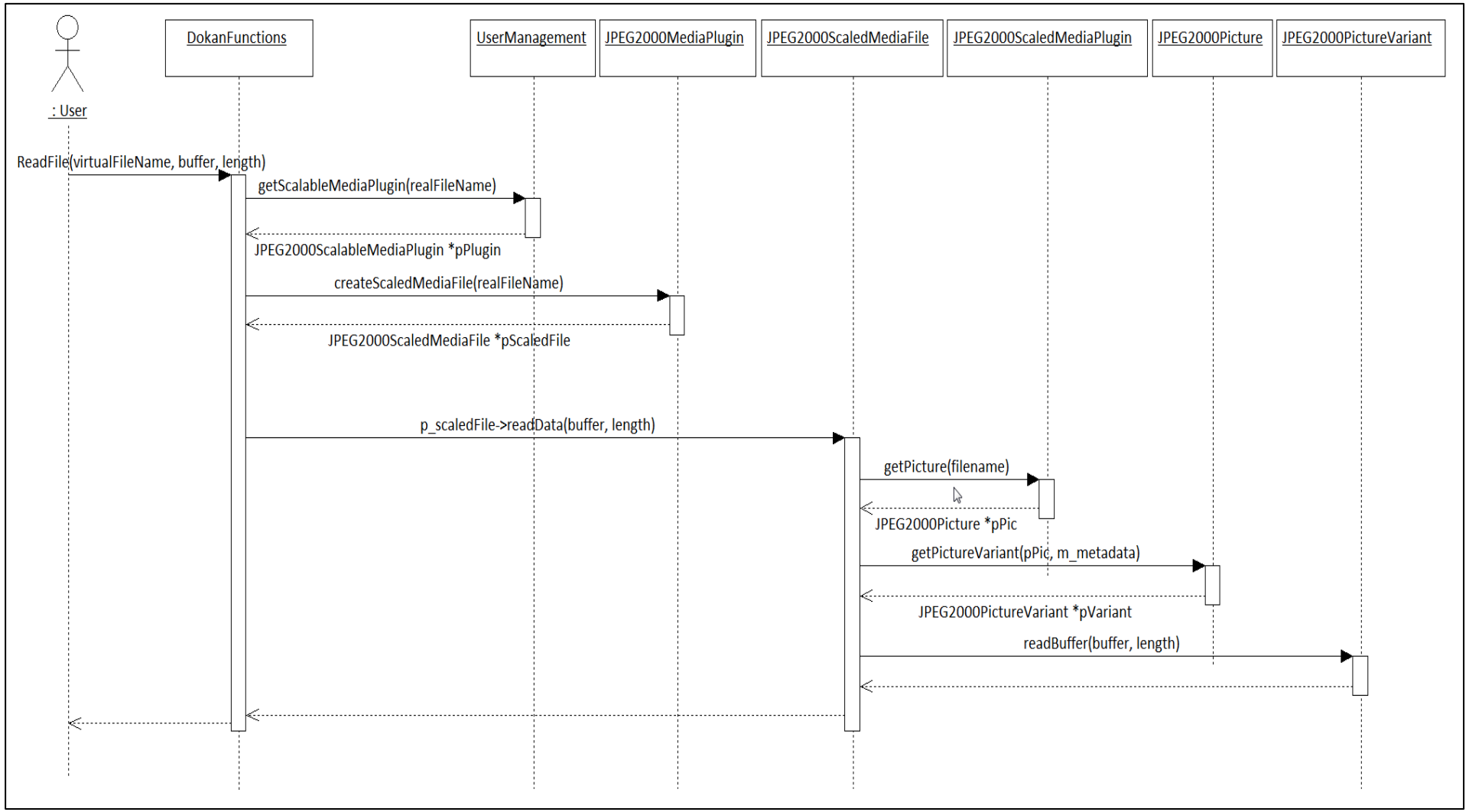

Abbildung 58: Sequenzdiagramm "Read File" im virtuellen Dateisystem für skalierbare Medien

D. Automatenentwicklung zum Parsen definierter Zusatzparameter

Eine übliche Pfadangabe inkl. einer Gruppierung ist in Abbildung 59 dargestellt. Der Automat muss später in der Lage sein, vorgeschaltete Hierarchiestufen — also Ordner, die nicht parametriert werden können — zu überlesen. Anschließend muss die Unterverzeichnisparametrierung geparst werden, bevor die Parameter für eine Datei aus dem Dateipfad gelesen werden können.

Für die Entwicklung des Parsers werden zudem zwei Wertebereiche definiert: Der Wertebereich *P* beinhaltet dabei alle Zeichen bis auf die benötigten Zeichen für die Parametrisierungsfunktionalität im Dateisystem, namentlich der *Backslash* und das Semikolon. Der Wertebereich *C* beinhaltet alle alphanumerischen Zeichen, also A-Z, a-z sowie 0-9 (vgl. Tabelle 15).

Zunächst sollen die Zustände und Übergänge des Automaten für die Unterverzeichnisparametrierung definiert werden. Abbildung 60 zeigt die entsprechenden Prozessschritte. Die in Abbildung 59 gezeigten vorgeschalteten Hierarchiestufen werden durch das Konstrukt *UV_NAME* sowie *UV_NAME_REST* verarbeitet. Dabei wird deutlich, dass es beliebig viele vorgeschaltete Ordner geben kann, bevor die Unterverzeichnisparametrierung beginnen kann. Hierbei ist anzumerken, dass die Abbildungen zu den Automatenzuständen aus Gründen der Übersichtlichkeit ohne die Fehlerbehandlung dargestellt sind. Der Übergang zum Parsen optionaler Parameter erfolgt durch das Semikolon, wodurch ein Übergang von *UV_NAME_REST* zu *UV_PARA*M ermöglicht wird. Anschließend wird der aktuelle Parametername ausgelesen. Handelt es sich bei dem Parameter um ein Flag (s.o.) kann der Vorgang potenziell beendet werden, wenn nach dem Flag keine weiteren Eingangszeichen vorhanden sind. Handelt es sich jedoch um einen Parameter mit anschließender Werteübergabe wie in Abbildung 59, wird durch das Sonderzeichen „=“ ein Übergang vom Zustand *UV_PARAM_REST* zu *UV_WERT* erreicht. Nachdem der Wert komplett ausgelesen ist, kann potenziell ein weiterer Parameter folgen — hierzu existiert einen Übergang von UV_WERT_REST zu UV_PARAM, der durch ein weiteres Semikolon ausgelöst wird. Anzumerken ist hier, dass als Parametername und Parameterwert alphanumerische Zeichen zugelassen sind.

In Abbildung 60 bleibt offen, wie der Übergang von der Unterverzeichnisparametrierung zur Dateiparametrierung erfolgt. Dieser Schritt wird später gezeigt. Zuvor zeigt Abbildung 61 die nötigen Zustände des endlichen Automaten für die Dateiparametrierung – zur besseren Übersichtlichkeit ebenfalls ohne Fehlerbehandlung. Der Vorgang des Parsens der optionalen

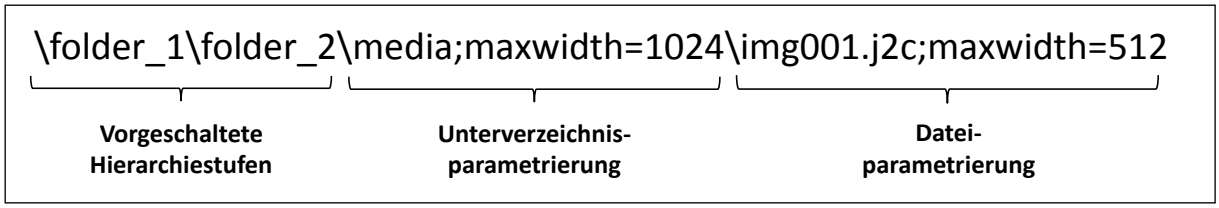

Abbildung 59: Beispielpfadangabe inkl. Parametrierung des Unterverzeichnisses und der Datei

Tabelle 15: Wertebereichsdefinition für C, P und ε

Name	Beschreibung
C	Alle Zeichen ohne \ und ;
P	Zeichen A-Z und a-z sowie die Ziffern 0-9
ε	Keine weiteren Zeichen vorhanden

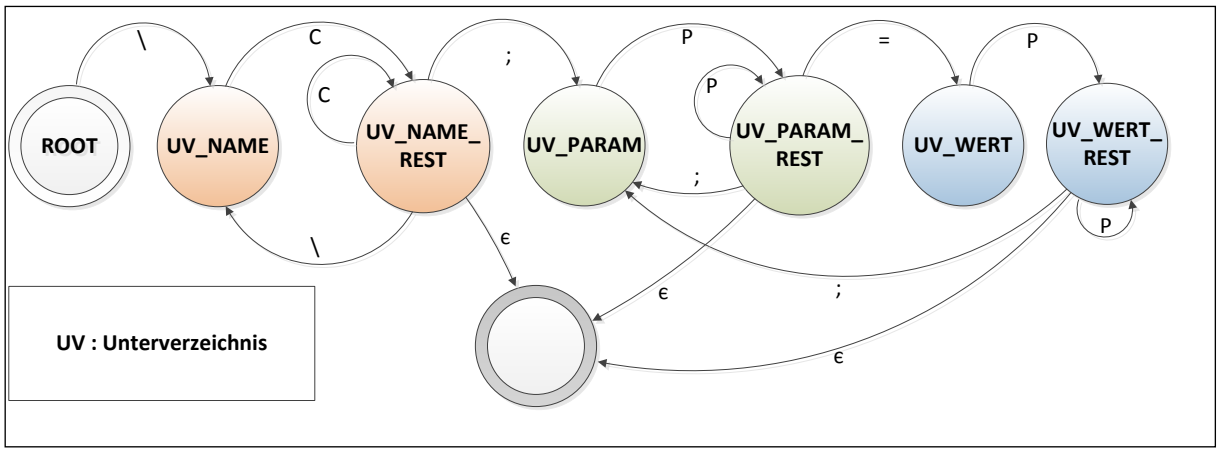

Abbildung 60: Zustände und Übergänge des endlichen Automaten beim Parsen der Unterverzeichnisparametrierung ohne Fehlerbehandlung

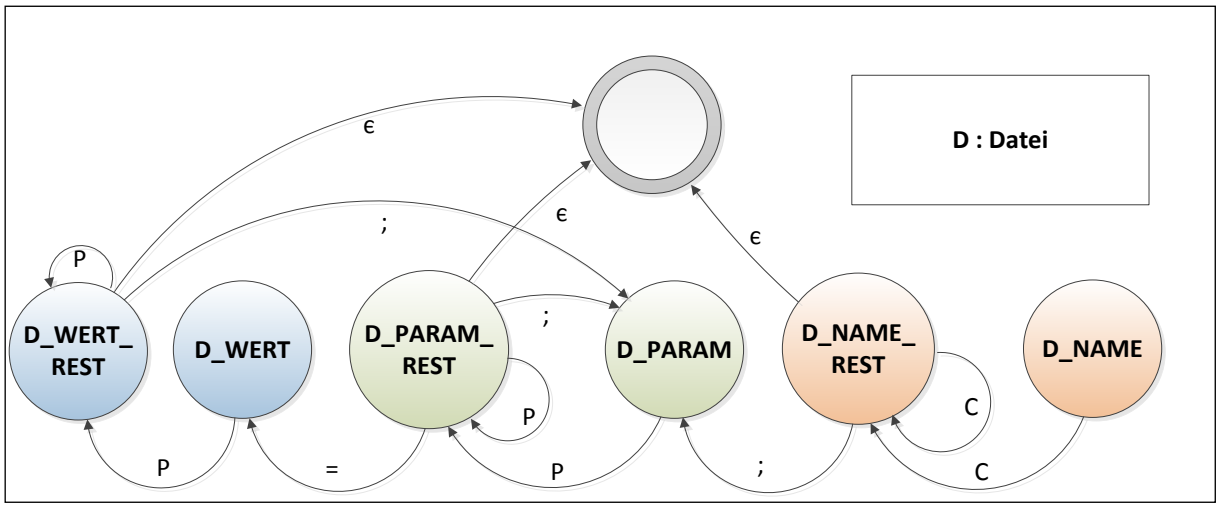

Abbildung 61: Zustände und Übergänge des endlichen Automaten beim Parsen der Dateiparametrierung ohne Fehlerbehandlung

der optionalen Parameter erfolgt dabei ähnlich dem bereits geschilderten Verfahren im Verzeichnisnamen. Die logische Anordnung ist in Abbildung 61 von rechts nach links

angeordnet. Dabei wird zunächst der Dateiname geparst. Der Zustand *D_NAME_REST* kann hier allerdings nicht durch einen *Backslash* in den Zustand *D_NAME* wechseln, wie das beim Zustand *UV_NAME_REST* und *UV_NAME* (vgl. Abbildung 60 der Fall ist. Das ist dadurch begründet, dass sich der Automat bereits in dem Zustand befindet, in dem er die Dateiparametrierung parst. Daraus folgt, dass optionale Parameter für das Unterverzeichnis bereits ermittelt wurden und das *Backslash*-Symbol nicht mehr zulässig ist. Die üblichen Zustände und Übergange sind analog zum Verfahren beim Parsen der optionalen Unterverzeichnisparameter.

Zuletzt bleibt noch zu klären, an welchen Stellen ein Übergang vom Parsen der Unterverzeichnisparameter zum Parsen der Dateiparameter erfolgen kann. Hierzu zeigt Abbildung 62 eine Zusammenfassung der bereits gezeigten Automatenteile. Neben der Fehlerbehandlung wurde hier auch die Endbehandlung ausgeblendet, um die Übersichtlichkeit zu erhöhen. Die Endbehandlung tritt immer dann ein, wenn kein weiteres Symbol mehr gelesen werden kann und ist in den Abbildung 60 und Abbildung 61 vollständig enthalten. Es ist zu erkennen, dass ein Übergang vom Parsen der Unterverzeichnisparameter zum Parsen der Dateiparameter aus den Zuständen erfolgen kann, in denen entweder ein Flag (*UV_PARAM_REST*) oder ein Parameterwert (*UV_WERT_REST*) komplett eingelesen wurde. Einen Übergang vom Parsen der Dateiparameter zu den Unterverzeichnisparametern existiert nicht.

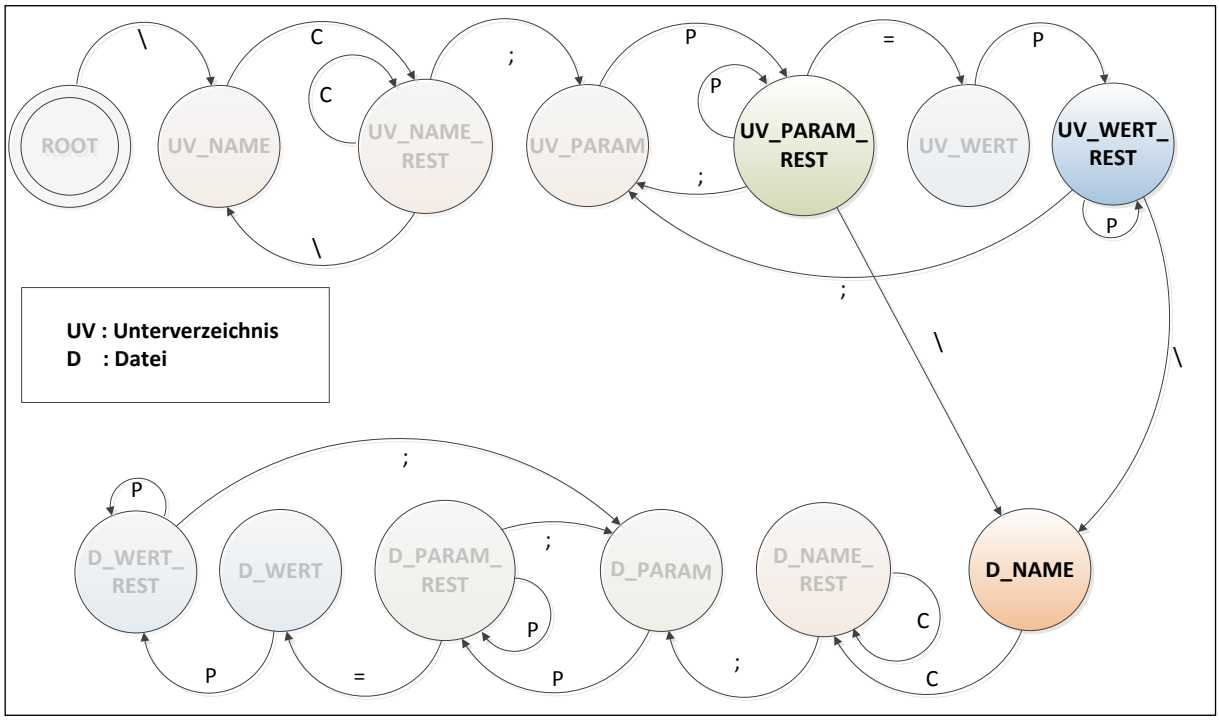

Abbildung 62: Übergänge des endlichen Automaten zwischen Unterverzeichnisparametrierung und Dateiparametrierung ohne Endzustände und ohne Fehlerbehandlung