Ruprecht-Karls Universität Heidelberg
Institut für Informatik
Abteilung Parallele und Verteilte Systeme

Bachelorarbeit

Container-Archiv-Format für wahlfreien effizienten Zugriff auf Dateien

Name: Hendrik Heinrich
Matrikelnummer: 2400353
Betreuer: Prof. Dr. Thomas Ludwig
Dipl. Ing. Rolf Kabbe
Abgabe Datum: 30.09.2007

Kurzfassung

Diese Arbeit entsteht in Zusammenarbeit mit unserem Forschungspartner Deutsches Krebsforschungszentrum (DKFZ) Heidelberg.

Ausgangspunkt der Arbeit ist eine große Anzahl verhältnismäßig kleiner Dateien, welche nur für den lesenden Zugriff verfügbar sein sollen. Diese entstehen am DKFZ durch Mikroskopie-Experimente, welche zur Zeit bereits 20 Millionen Bilddateien für die spätere Auswertung geliefert haben.

Allein das Anzeigen eines Verzeichnisinhalts mit *ls* oder *ls –l* kann, wegen der großen Anzahl der Bilddateien, Minuten in Anspruch nehmen. Als Fazit eines Praktikums der Autoren Michael Kuhn und Christian Lohse, welches die Evaluierung paralleler Dateisysteme zum Thema hatte, stellte sich heraus, dass man der übergroßen Zahl an Dateien mit einer Container-Archiv-Lösung Herr werden könnte. Das Problem liegt hier im speziellen bei den Leistungsengpässen, die durch Dateien entstehen, welche ein Verzeichnis repräsentieren. Dies dient für die Arbeit als Motivation und Berechtigung.

Als Ergebnis dieser Arbeit entstand sowohl die Definition eines Container-Archiv-Formats, als auch ein Paket von Kommandozeilenprogrammen, sowie eine API, mit deren Hilfe sich Containerdateien erstellen und verwenden lassen. Die Kommandozeilenprogramme sowie die API belegen, verifiziert durch Tests, dass die Verwendung eines solchen Containers Leistungsvorteile bringt. Hierbei hat sich die Verwendung der Containerdatei gegenüber der Verwendung von gängigen Archiv-Formaten und Linux-Dateisystemen bewiesen.

Danksagung

Vielen Dank allen, die an der Entstehung dieser Arbeit beteiligt waren. An erster Stellen den Betreuern Prof. Dr. T. Ludwig und Dipl. Ing. R. Kabbe.

Ein besonderer Dank gilt den Mitgliedern der Arbeitsgruppe um Prof. Dr. T. Ludwig für die immer hilfreiche Unterstützung.

Hier speziell Michael Kuhn und Julian Kunkel, die beim konzeptionellen Entwurf, viel Zeit und Muße investierten.

Inhaltsverzeichnis

1 Einleitung 9
1.1 Zielsetzung 10
1.2 Aufbau der Arbeit 10
2 Problemstellung 13
3 Containerformat 19
3.1 Diskussion der Vorgaben und Einschränkungen 19
3.2 Vergleichbare Container-Archiv-Formate 20
3.3 Definition des Containerformats 21
3.4 Definition/Herleitung resultierender Zusammenhänge 23
3.4.1 Begriffsdefinition 23
3.4.2 Reservierte Größen in Byte 24
3.4.3 Resultierende Abhängigkeiten 25
4 Schnittstellendefinition (API) 29
4.1 Kommandozeilenorientierte Schnittstellen 29
4.1.1 Containererstellung 30
4.1.1.1 *ctmk*-Implementierung 30
4.1.2 Auflistung des Containerinhalts 32
4.1.2.1 *ctls*-Implementierung 32
4.1.3 Kopieren enthaltener Datei 34
4.1.3.1 *ctfcp*-Implementierung 35
4.2 Definition der Anwendungs-API 37
5 Evaluation 43
5.1 Testumgebung 43
5.2 Test: Erstellung 43
5.2.1 Testablauf: Erstellung 45
5.2.2 Auswertung: Erstellung 45
5.3 Test: Auflistung des Inhalts 48
5.3.1 Testablauf: Auflistung des Inhalts 48
5.3.2 Auswertung: Auflistung des Inhalts 49

5.4 Test: Dateizugriff....54
5.4.1 Testablauf: Dateizugriff....54
5.4.2 Auswertung: Dateizugriff....55
Zusammenfassung / Bewertung....61
Ausblick....63
Literaturverzeichnis....65
Anhang A....67
Testprotokolle....67
Testergebnistabellen....85
Testergebnisdiagramme....89
Anhang B....93
Quelltext *libct.h*....93
Quelltext *libct-internal.h*....95
Quelltext *ctest.sh*....97
Quelltext *create_testfiles.sh*....103

1 Einleitung

Die Effizienz von Dateisystemen bezüglich Zugriffszeiten spielt besonderes im universitären Umfeld ein wichtige Rolle. Da hier übergroße Mengen von Messdaten zur späteren Auswertung entstehen, müssen die eingesetzten Verfahren zur digitalen Verwaltung und Archivierung von Dateien besonders effizient sein. Schon jetzt ist es so, dass jeden Tag mehr Informationen gespeichert werden als in der gesamten Geschichte der Menschheit zuvor. Dabei stellt sich bereits die Frage, ob und wann genügend Zeit zur Verfügung steht, die gesammelten Daten auszuwerten. Für die effiziente Verarbeitung großvolumiger Datenmengen stehen die Lösungen aus dem Bereich des Cluster-Computing und für die effizienten Übertragung die modernen Netzwerklösungen zur Verfügung. Auch moderne Dateisysteme werden bezüglich des Zugriffs auf die Daten immer effizienter. Die bei der Entwicklung von Dateisystemen zwangsweise eingegangenen Kompromisse jedoch, können auch negative Auswirkungen auf die Zugriffseffizienz entwickeln. Ein Beispiel dafür sind die Leistungsmängel, die auch bei modernen Dateisystemen beim Handhaben großer Dateien und Verzeichnisse auftreten.

Bei unserem Forschungspartner, dem Deutschen Krebsforschungszentrum (DKFZ) entstehen große Mengen an Bilddateien im Zusammenhang mit Mikroskopie-Experimenten. Dabei werden drei Monate lang alle 30 Minuten 384 Bilder aufgenommen und zur späteren Verarbeitung digital gespeichert. Das entspricht einem Datenvolumen von mehr als 3 TByte in mehr als zwei Millionen Dateien.

Da Dateisysteme eine Vielzahl von Datei-Typen und damit auch alle denkbaren Dateigrößen unterstützen müssen, ist die Implementierung eines idealen Dateisystems unmöglich. Die Kompromisse, die bei der Konzeption von Dateisystemen eingegangen werden, und deren Umsetzung führt im allgemeinen dazu, dass nicht für jeden Anwendungsfall die bestmögliche Leistung erreicht wird. Besonders bei großen Dateien oder großen Verzeichnissen ist es nicht ungewöhnlich, dass sich die Zugriffszeiten pro Datei im Vergleich zu kleinen Dateien und Verzeichnissen verdoppeln. Dadurch kann das einfache Auflisten des Verzeichnisinhalts mehrere Minuten in Anspruch nehmen.

Während eines Praktikums wurden Dateisysteme dahingehend untersucht, ab welcher Menge von Dateien sie Engpässe in der Leitung aufweisen [KL07]. Da hier die Leistungsfähigkeit der Dateisysteme mit großen Anzahlen von Dateien kleiner Größe getestet wurde, sind die Leistungseinbrüche in der Verwaltung der Dateien zu vermuten.

Zur Verwaltung von Dateien und Verzeichnissen sind nicht nur die Nutzdaten also der Dateiinhalt nötig, sondern auch Informationen über die Datei oder das Verzeichnis selbst. Diese Informationen werden Metadaten genannt.

In dieser Arbeit wird evaluiert ob, und in welcher Größenordnung eine Minimierung der Metadatenzugriffe Leistungsvorteile gegenüber der Verwendung von Standard-UNIX-Linux-Dateisystemen bringt. Dafür wird ein Format für Containerdateien definiert, so dass das Dateisystem für den Inhalt keinerlei Metadaten zu verwalten hat. Dieses Format muss

ein eigenes Konzept zur Verwaltung des Containerinhalts implementieren. Da die Komprimierung der eigentlichen Nutzdaten, in dieser Arbeit keine Betrachtung finden wird, sind die Steigerung der Zugriffsleistung vor allem durch Steigerung der Speichereffizienz durch Verringerung, der Metainformationen und durch effiziente Verwaltungsmechanismen des Inhalts zu erreichen.

Zur Erreichung der Leistungssteigerung werden folgende Ziele definiert.

1.1 Zielsetzung

Ziel dieser Arbeit ist es, den Zeitaufwand für die Auflistung von Verzeichnisinhalten durch Verringerung der Metadatenzugriffe zu minimieren. Dabei wird von großen Verzeichnissen ausgegangen, also Verzeichnissen mit einer hohen Anzahl von Dateien, um den Anwendungsfall am DKFZ gut zu approximieren. Dazu wird ein Container definiert, in welchem die Dateien gehalten werden. Das Containerformat implementiert ein effizientes Konzept zur Verwaltung des Containerinhalts. Durch das Einbringen von zusätzlichem impliziten Wissen über den Inhalt eines Containers, ist der Verwaltungsaufwand zu minimieren.

Für diesen Container sollen die nach POSIX standardisierten Zugriffsverfahren weitgehend gelten. Dabei wird der eigentliche Dateizugriff auf eine im Container befindlichen Datei wahlfrei sein und möglichst keine zusätzliche Zeit in Anspruch nehmen.

Die Erstellung einer entsprechenden API zur Verwendung einer Containerdatei ist ebenfalls Ziel dieser Arbeit. Abschließend ist es Ziel, mit Tests zu evaluieren, welche Leistungsvorteile durch die Verwendung einer Containerdatei gegenüber gängigen Dateisystemen und Container-Archivformaten erreicht werden.

Da das Containerformat auch in anderen Anwendungsbereichen, außer dem in dieser Arbeit angeführten Verwendung finden können soll, ist die mögliche Erweiterbarkeit und Wiederverwendbarkeit bei der Entwicklung nicht zu vernachlässigen. Das Hauptaugenmerk liegt jedoch auf der grundsätzlichen Evaluierung, ob die Verwendung einer Container-(Archiv)-Lösung eine signifikante Leistungssteigerung für den hier zu Grunde liegenden Anwendungsfall bietet.

1.2 Aufbau der Arbeit

Die folgende Aufstellung liefert einen Überblick der Hauptpunkte dieser Arbeit:

- Betrachtung der Problemstellung

  Im Kapitel Problemstellung wird untersucht, wo die Mängel in der Leistung bei der Handhabung großer Dateien und Verzeichnisse ihre Ursachen haben. Dazu wird ein Einblick in die Strukturen des Virtual File Systems VFS und des Ext3-Dateisystems gewährt, um sich der Problematik anzunähern.

- Definition des Containerformats

  Bevor in diesem Kapitel das Format der Containerdatei definiert wird, erfolgt eine Betrachtung des Aufbaus des TAR-Archiv-Formats, als gängigem Vertreter einer Container-Archiv-Lösung. Zuvor werden die, für diese Arbeit geltenden Vorgaben und Einschränkungen bezüglich des Containerformatsdiskutiert.

- Schnittstellendefinition für den Zugriff auf Containerdateien

  Die für die Verwendung eines Containers nötigen Schnittstellen, auch API genannt, werden in diesem Kapitel definiert. Da nicht nur aus Anwendungen heraus auf Containerdateien zugegriffen werden soll, werden hier auch die Implementierungen der nötigen Kommandozeilenprogramme betrachtet. Diese bilden eine weitere API auf einer anderen Abstraktionsebene.

- Implementierung und Evaluation

  Zur Evaluation der Leistung werden in diesem Kapitel Test-Skripten, Testfälle und Testumgebungen definiert. Hier zeigt das definierte Containerformat seine Daseinsberechtigung gegenüber gängigen Dateisystemen und Container-Archiv-Formaten.

Anschließend folgt die Zusammenfassung und Bewertung der in der Arbeit vorgestellten Definitionen und Tests und es wird ein Ausblick auf zukünftige Fragestellungen im Zusammenhang mit dem Containerformat gewährt.

2 Problemstellung

In diesem Kapitel wird die Ursache der Leistungsengpässe untersucht, die im Zusammenhang mit großen Dateien und Verzeichnissen entstehen

Ein besonderes Merkmal von UNIX-Betriebssysteme wie Linux ist die einheitliche Darstellung des Gerätezugriffs auf Anwendungsebene durch das Konzept der Datei. Auf der Ebene des Betriebssystemkerns wird dies durch eine Zwischenschicht als virtuelles Dateisystem implementiert. Dieses so genannte Virtual File System VFS ist unter Linux im Kernelquellcode unter */usr/src/linux/fs* zu finden und in [GE05] dokumentiert. Eine detailliertere Beschreibung findet man in [BM00, Mau04, Lov05]. Das VFS ermöglicht nicht nur den transparenten Zugriff auf eine Vielzahl von Geräten, sondern abstrahiert auch vom jeweils vorliegenden Dateisystemen beim Zugriff auf Sekundär-Speichermedien. Damit lassen sich eine Vielzahl von Dateisystemen durch einheitliche Schnittstellen transparent unterstützen. Das VFS besteht aus einer Menge von Strukturen zur Verwaltung von Dateisystemobjekten, wie Dateien und Verzeichnisse. Zu jeder dieser Strukturen existieren weitere Strukturen, die die Operationen definieren, die auf Dateisystemobjekte angewandt werden können. In diesen Operationsstrukturen werden Zeiger auf Funktionen definiert. Die Implementation der Funktionen obliegt dem zugehörigen Dateisystem.

Im VFS ist ein Dateisystem durch einen *Superblock* eindeutig definitert. Er enthält die Metadaten des Dateisystems. Ein *Superblock*, der zum Beispiel das Dateisystem Ext3 charakterisiert, kann in mehreren Instanzen existieren, da innerhalb des Verzeichnisbaumes mehrere Ext3-Dateisysteme eingehängt sein können.

Zentrale Struktur des VFS ist der *Inode*, die Kurzform von Index-Knoten. Jede Datei wird durch einen *Inode* charakterisiert. Da im VFS alles eine Datei ist, sind auch Verzeichnisse Dateien, also durch einen *Inode* charakterisiert. In einer solchen *Inode*-Struktur befinden sich alle dateibezogenen Daten, die Metadaten des assoziierten Dateisystemobjekts. Dazu gehören die Zugriffsrechte, bei Dateien die Dateigröße, die Zeitstempel aber nicht der Datei- oder Verzeichnisname.

Zur Verwaltung der Objektnamen ist im VFS eine weitere Struktur namens *Dentry*, kurz für Directory-Entry definiert. Neben dem Hauptzweck, die Verbindung zwischen Dateinamen und *Inode* herzustellen, bilden die *Dentry*-Instanzen ein Netzwerk welches die Struktur des Dateisystems nachbildet. Dazu speichert jedes *Dentry* eine Liste *d_subdirs,* die für alle seine Unterverzeichnisse und Dateien *Dentry*-Objekte enthält. Zusätzlich wird eine weitere Liste *d_child* gespeichert, die die *Dentry*-Objekte zu den Eltern-Objekten enthält.

Der Zugang zu den eigentlichen Nutzdaten einer Datei erfolgt durch einen Zeiger der *Inode*-Struktur und einer weiteren Struktur *File*. Diese enthält einen Zeiger *f_pos*, der die aktuelle Position des Lese-/Schreib-Zeigers innerhalb der Daten einer Datei enthält. Eine solche *File*-Struktur wird immer dann erzeugt, wenn ein Prozess eine Datei zur Verwendung öffnet und existiert auch nur so lange die Datei geöffnet ist. Der *Superblock* eines

Dateisystems enthält eine Liste *s_list* aller geöffneten Dateien in Form einer Auflistung dieser *File*-Strukturen. Auch ein Prozess hält in seiner Task-Struktur, die hier nicht weiter erläutert werden soll, eine Struktur files in welcher er mit Hilfe von Zeigern, auch Datei-Deskriptoren genannt, alle zu ihm assoziierten *File*-Objekte verwaltet. Das *File*-Objekt repräsentiert damit die Sicht eines Prozesses auf ein offene Datei im Speicher.

Das VFS hat wesentlich mehr als die hier erwähnten Bestandteile, dennoch wird hier auf eine weitergehende Detaillierung verzichtet, um die Übersicht nicht zu behindern. Die folgende Abbildung zeigt die gerade beschriebenen Zusammenhänge.

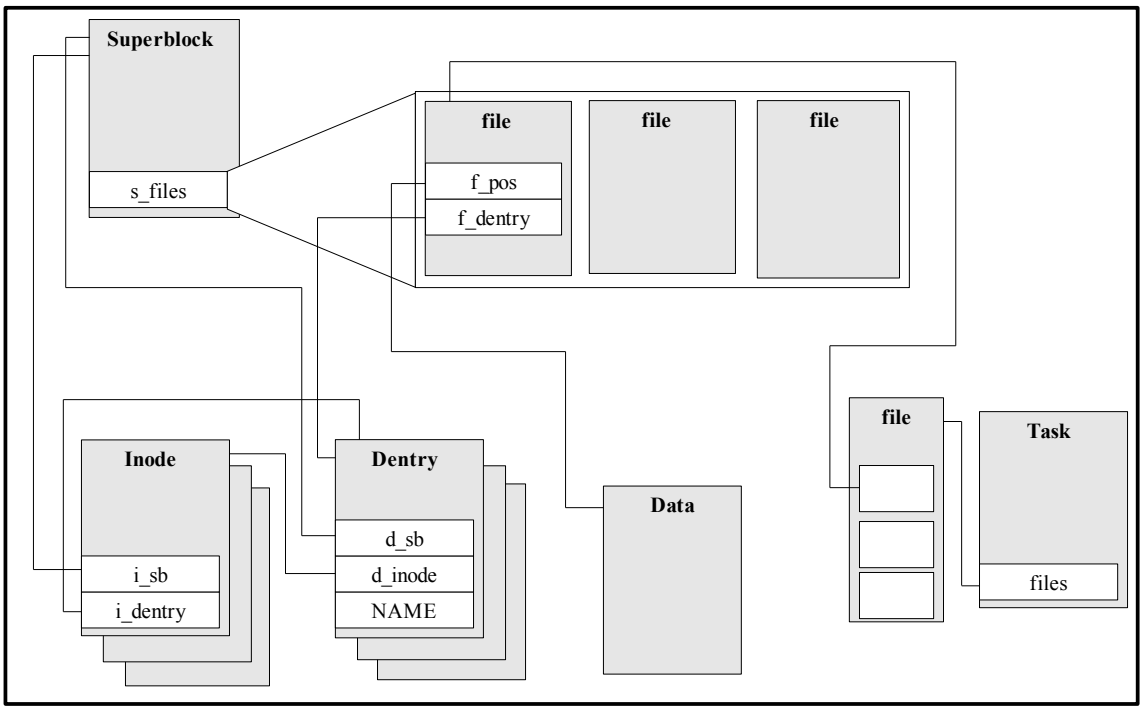


Abbildung 2.1: Virtual File System

Nachdem nun ein grober Überblick über die Datenstrukturen des VFS und deren Zusammenhängen besteht kann nun ein Beispiel-Szenario betrachtet werden. Es soll grob darstellen welche Informationen das VFS vom Dateisystem geliefert bekommen muss, um zum Beispiel einen Verzeichnisinhalt darstellen zu können. Dabei soll angenommen werden, dass das Verzeichnis *DIR* heißt, sich in */home* befindet und 1 Mio Dateien *file1 – file1000000* enthält.

Es werden also die *Inodes* der Dateien gesucht, da diese die Metadaten enthalten. Um die *Inodes* mit Hilfe der Dateinamen zu finden, werden die *Dentrys* benötigt, da nur sie den Zusammenhang zwischen Dateinamen und *Inode* im VFS repräsentieren. Zunächst betrachten wir nur die erste Datei. Dafür muss das VFS für *file1* in */home/DIR* eine *Dentry*-Struktur angelegen und mit den entsprechenden Informationen füllen. Das Gleiche gilt für den assoziierten *Inode*. Der *Lookup*-Mechanismus des VFS liefert die gewünschten Informationen indem er alle Verzeichnisse des Pfades, oder genauer deren assoziierte *Inodes* auf Existenz und Berechtigung prüft. In diesem Beispiel sind die zu untersuchenden Ver-

zeichnisse */*, *home* und *DIR*. Als Ergebnis liegt die gewünschte *Dentry*-Struktur zu *file1* vor. Bleibt noch, den assoziierten *Inode* zu referenzieren, um das Lesen der Metadaten der ersten Datei zu ermöglichen.

Während es im Falle der ersten Datei durchaus möglich ist, dass es noch keine Caching-Effekte gab, um davon zu profitieren, sind für jede weiter Datei zumindest die *Dentry*-Strukturen für den Pfad im *Dentry*-Cache des Betriebssystemkerns. Damit wird der Zeitaufwand für den *Lookup*-Mechanismus für die folgenden Dateien erheblich beschleunigt, da die relevanten Daten schon im schnellen Hauptspeicher liegen und daher weniger Zugriffe auf langsame Blockspeichermedien notwendig sind. Damit lässt sich auch für jede weitere Datei effizient eine *Dentry*-Struktur mit den nötigen Daten des Dateisystems belegen. Für diese, als auch für jede weitere Datei muss noch die entsprechende VFS-*Inode*-Struktur mit Daten, die das darunter liegende Dateisystem zu liefern hat, gefüllt werden.

Auch wenn der soeben gelieferte Ablauf eine starke Vereinfachung der tatsächlich Vorgänge darstellt lässt er doch erkennen wo die Leistungsengpässe ihre Ursache haben. Sobald die Datenstrukturen, die das VFS für die Datei- und Verzeichnisverwaltung benötigt, im Hauptspeicher, also im RAM, verfügbar sind kann es seine Aufgabe effizient erfüllen. Daraus lässt sich schließen, dass die Beschaffung der Daten, die dem darunter liegenden Dateisystem obliegt, für die Leistungsengpässe ursächlich ist.

Um sich dem Problem noch weiter anzunähern sollen die Datenstrukturen eines Dateisystems auf einem Blockspeichermedium untersucht werden. Als Vertreter eines Dateisystems wird hier Ext3 betrachtet, da es in seiner Struktur dem VFS sehr ähnlich, und damit gut vergleichbar ist. Ext3 gehört zu den Standard-Dateisystemen auf einem Linux-System. Es stimmt bis auf hier nicht relevante Erweiterungen, wie das Journaling, in seinen Strukturen auf der Festplatte mit Ext2 überein [Ext3]. Daher kann für die folgenden Betrachtungen auch Ext3 als Synonym für Ext2 verstanden werden [CTT].

Das grundsätzliche Problem, dass Dateisystemstrukturen auf Blockspeichermedien haben ist der folgende Kompromiss, dessen Umsetzung sich negativ vor allem auf große Dateien auswirkt, die ein Verzeichnis repräsentieren. Einerseits gilt es die räumliche Nähe zwischen Eltern- und Kindverzeichnissen oder speziell deren Metadaten zu minimieren und andererseits die räumliche Nähe der Metadaten und den Datenblöcken eines Dateisystemobjekts zu minimieren. Übersteigt die räumliche Entfernung in einem der beiden Fälle eine gewisse Größe greifen durch das Betriebssystem implementierte read-ahead-Mechanismen nicht mehr. Daraus resultiert die Notwendigkeit für zusätzliche Lesezugriffe auf das Blockspeichermedium, die sehr zeitintensiv sind. Diese Leistungsengpässe werden noch verstärkt, wenn diese Zugriffe zusätzlich mit einer Neupositionierung der Leseköpfe verbunden sind.

Um die physikalische Nähe von Meta- und Nutzdaten eines Dateisystemobjekts zu gewährleisten, fasst das Ext3-Dateisystem die zur Verfügung stehenden Blöcke eines Blockspeichermediums zu Blockgruppen zusammen. Jede dieser Blockgruppen stellt einen Block für den *Superblock* des Dateisystems, *k* Blöcke für Blockgruppen-Deskriptoren, einen Block für eine Daten-Bitmap, einen Block für eine *Inode*-Bitmap, *n* Blöcke für eine *Inode*-Tabelle und *m* Blöcke für Daten bereit. Der *Superblock* enthält die Metadaten des Dateisystems. Da ohne ihn das Dateisystem unbrauchbar ist, wird er redundant gespei-

chert. Jede Blockgruppe enthält einen Gruppendeskriptor für alle Blockgruppen des Dateisystems, mit deren Hilfe Informationen, wie die Anzahl freier Blöcke und *Inodes* jeder Blockgruppe wiedergegeben werden. In der Daten-Bitmap definiert jedes gesetzte Bit einen benutzten Datenblock.

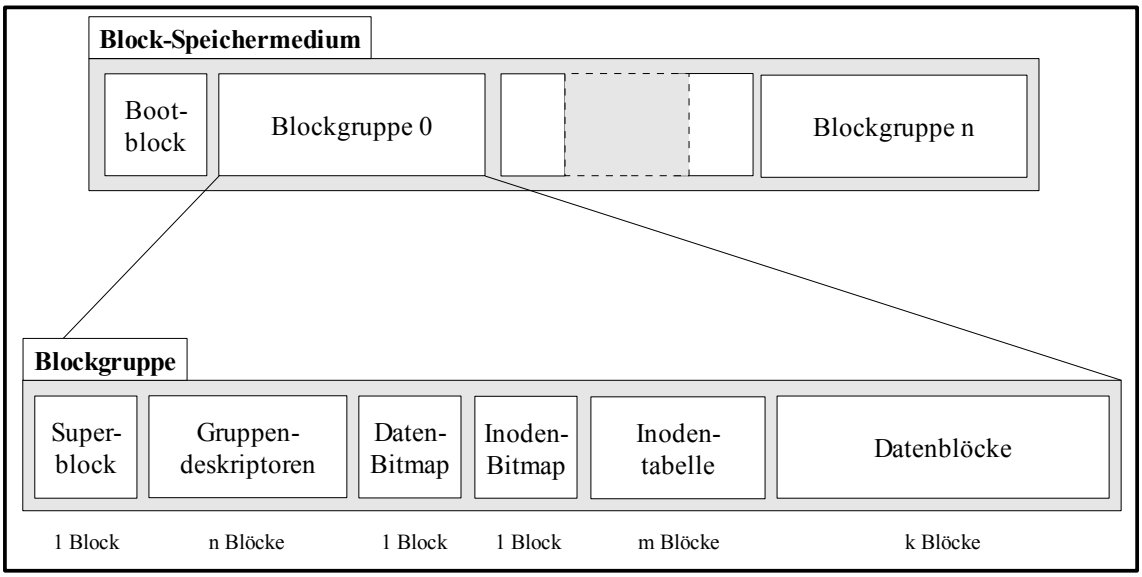


Abbildung 2.2: Ext3 Blockspeichermedien-Struktur

Äquivalent dazu definiert jedes gesetzte Bit in der *Inode*-Bitmap ein genutzten *Inode*. Die *Inode*-Tabelle nimmt alle *Inodes* der Blockgruppe auf und die Datenblöcke halten die Nutzdaten eines Dateisystemobjekts. Damit wird erreicht, dass eine Teilmenge der vorhandenen *Inodes* physikalisch nah einer Teilmenge der vorhandenen Datenblöcke liegt.

Eine Datei ist in dieser Struktur folgendermaßen abgelegt. Es ist für die Datei eine *Inode* belegt, die deren Metadaten enthält. Diese *Inode* referenziert so viele Datenblöcke wie nötig sind, um die Nutzdaten der Datei aufzunehmen. Dazu stellt eine *Inode* 12 + 3 Zeiger bereit.

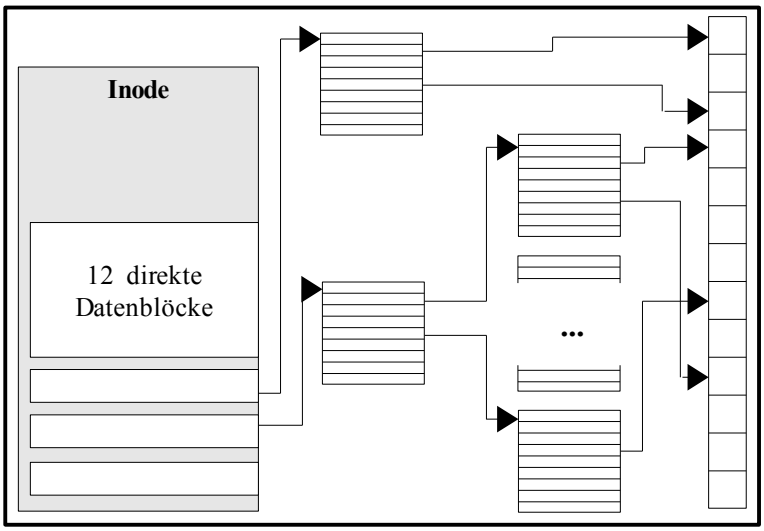


Abbildung 2.3: Ext3 Indirektion

Die ersten zwölf Zeiger referenzieren direkt zwölf Datenblöcke. Der Zeiger 13 dient der einstufigen, 14 der zweistufigen und 15 der dreistufigen Indirektion. Bei der Indirektion werden Datenblöcke referenziert, die wiederum Zeiger auf Datenblöcke enthalten. Damit lassen sich bei einer Standard-Blockgröße von 4096 Byte Dateien bis zu einer Größe von 2 TByte verwalten [Wik].

Verzeichnisse sind, wie schon erwähnt, auch Dateien und werden damit ebenfalls durch *Inodes* repräsentiert. In den Datenblöcken eines Verzeichnisses hingegen werden *Dirent*-Strukturen gehalten. Sie sind im Ext3-Dateisystem nicht so ausführlich, enthalten also weniger Datenfelder als die aus dem VFS bekannten *Dentry*-Strukturen, dienen jedoch dem selben Zweck. Sie bilden die Verzeichnisstrukturen des Dateisystems auf ein Blockspeichermedium ab. Dazu enthalten sie den Namen des Kind-Objektes und einen Verweis auf den assoziierten *Inode*. Wenn ein so dargestelltes Verzeichnis eine große Anzahl von Kind-Objekten hat kann es auch hier die Notwendigkeit für Indirektion bei dessen Datenblock-Referenzierung kommen.

Auf weitergehende Erläuterung der Details wird hier verzichtet, da sich aus den benannten Zusammenhängen, das Problem der Leistungsmängel bei großen Dateien und Verzeichnissen darstellen lässt. Dazu soll die Betrachtung einer großen Datei und eines großen Verzeichnisse dienen. Im speziellen Fall, des Ext3-Dateisystems kann eine Datei, wegen der gezeigten Indirektion, maximal 2 TByte groß sein. Dateien dieser Größe lassen sich bei einer Blockgröße von 4096 Byte verwalten. Um die 2 TByte Nutzdaten auf dem Blockspeichermedium abzulegen müssen mehr als 500 Millionen Datenblöcke referenziert werden. Um effizient auf eine solche Dateien zugreifen zu können, müssen die Datenblöcke möglichst zusammenhängend auf dem Blockspeichermedium abgelegt sein. Die Aufteilung der verfügbaren Blöcke in Blockgruppen wie im Ext3-Dateisystem, verhindert dies aber. Da jede Blockgruppe nur einen Block für die Daten-Bitmap speichert, stehen aber nur maximal $4096\ Byte \cdot 8 = 32768\ Bit$ zur Verfügung, um den Zustand eines Datenblocks wiederzugeben. Daraus folgt wiederum, dass Datenblöcke anderer Blockgruppen referenziert werden müssen. Diese Fragmentierung der Datei hat zur Folge, dass das Blockspeichermedium, zum Beispiel eine Festplatte, zusätzliche Positionierungen der Lese-/Schreib-Köpfe vornehmen muss.

Bei einem großen Verzeichnis, wie dem oben erwähnten *DIR*, mit einer Million Kind-Objekten, tritt das gleiche Problem auf. Der Idealzustand wäre, dass das *Inode* des Verzeichnisses und die *Inodes* der Kind-Objekte innerhalb einer Blockgruppe platz fänden. Die Anzahl der *Inodes* innerhalb einer Blockgruppe ist aber begrenzt. Daher entsteht auch bei Verzeichnissen die Notwendigkeit auf *Inodes* anderer Blockgruppen zurückzugreifen. Auch dadurch bedingt, dass die Datenblöcke der Kind-Objekte möglichst nah bei den den *Inodes* der Kind-Objekte liegen sollen.

Zusammenfassend wird festgehalten, dass bei großen Dateien und damit auch bei großen Verzeichnissen folgende Probleme auftreten. Die Anzahl der Datenblöcke innerhalb einer Blockgruppe zur Aufnahme von Nutzdaten einer Datei reicht bei großen Dateien nicht aus. Das selbe Problem tritt bei Verzeichnissen auf die eine große Zahl an Verweisen auf Kind-Objekte in ihren Datenblöcken ablegen. Es ist auch ein Problem wenn die Anzahl

der *Inodes* nicht ausreicht, um zu jedem *Dirent* eines Verzeichnisses einen *Inode* in der selben Blockgruppe zu referenzieren. Als Folge dieser Problematik werden Datenblöcke und *Inodes* anderer Blockgruppen referenziert. Damit wird aber deren physikalische Entfernung zueinander größer. Es kommt daher ab einer gewissen Größe von Dateien und Verzeichnissen dazu, dass asynchrone Read-Ahead-Mechanismen die der Betriebssystemkern verwendet, um die Anzahl von E/A-Operationen zu minimieren nicht mehr greifen. Das heißt es werden zusätzliche E/A-Operationen nötig. Da diese Operationen zu den zeitlich teuersten Operationen in einem modernen Computer-System gehören, liegt genau hier die Ursache für Leistungsengpässe, die im Zusammenhang mit großen Dateien entstehen, die ein Verzeichnis repräsentieren.

3 Containerformat

Bevor in diesem Kapitel das Format der Containerdatei definiert wird, erfolgt die Betrachtung wie das TAR-Archiv-Format, als gängiger Vertreter einer Container-Archiv-Lösung aufgebaut ist. Einleitend werden die für diese Arbeit geltenden Vorgaben und Einschränkungen bezüglich des Containerformats diskutiert.

3.1 Diskussion der Vorgaben und Einschränkungen

Das Hauptziel, dass durch die Verwendung einer Containerdatei realisiert werden soll ist Leistungsgewinn bezüglich der Zeit. Das Format der Containerdatei ist also grundsätzlich diesem Ziel unterzuordnen.

Ein Weg, dieses Ziel zu erreichen ist die Zusatzlast, die eine Containerdatei mit sich bringt, minimal zu halten. Mit Zusatzlast sind hier alle zur Verwaltung des Containerinhalts notwendig Informationen gemeint, die zusätzlich zu den eigentlichen Nutzdaten anfallen. Als Teil der Zusatzlast werden der mögliche Header und das mögliche Inhaltsverzeichnis der Containerdatei angesehen. Wobei der Header eine Möglichkeit ist, initial benötigte Information über die Struktur der Containerdatei zur Verfügung zu stellen. Ganz ohne Zusatzlast ist eine solche Containerdatei nicht zu realisieren, da ja mindestens eine Menge von Verweisen auf die enthaltenen Dateien vorhanden sei muss, um sinnvoll auf den Inhalt der Containerdatei zugreifen zu können. Denkbar ist hier die Verwendung von Zeigern oder die Verwendung von Offsets. Wobei der Offset natürlich nur Sinn macht wenn die Zusatzlast eine möglichst statische Größe besitzt und/oder sinnvoll in der Containerdatei positioniert ist. Dabei ist vor allem die Frage zu beantworten, wie viele dateibezogene Informationen nötig sind, um eine eindeutige Identifikation einer Datei zu gewährleisten. Weniger ist hier mehr.

Eine weitere Einschränkung ist, dass die in der Containerdatei enthaltenen Dateien als Besitz dessen Benutzers angenommen werden, der die Containerdatei besitzt. So kann an dieser Stelle die Zusatzlast minimiert werden, da die Berechtigung für die Dateien nicht in der Containerdatei gespeichert werden müssen, sondern sich aus den Metadaten des Containers extrahieren lassen.

Zusätzlich ist es Vorgabe, dass die Containerdatei nur für den lesenden Zugriff konzipiert wird, da dass dem aus der Praxis resultierenden Gebrauch am nähesten kommt. Daraus ergibt sich aber auch, dass mit nachträglichen Änderungen des Inhalts nicht umgegangen werden muss. Dies kann wiederumzur Minimierung der Zusatzlast dienen.

Abschließend ist es noch Vorgabe, dass nicht nur der sequenzielle Zugriff möglichst effizient ist, sondern auch der wahlfreie. Das heißt, dass das Inhaltsverzeichnis nicht nur in Form einer Listen-Struktur zugreifbar sein sollte, sondern zusätzlich noch als Baum-

Struktur, da das Durchsuchen von Bäumen eine wesentlich geringere Komplexität als das Durchsuchen von Listen hat. Auf der Anderen Seite würde die Verwendung von Liste und Baum zusätzliche Last bedeuten. Hier ist also abzuwägen, ob die Verwendung beider Strukturen unumgänglich ist.

3.2 Vergleichbare Container-Archiv-Formate

Aus der Vielzahl bereits existierenden Container-Formate wird hier ein Vertreter betrachtet, der sich ebenfalls mit der Archivierung und damit Kapselung von Dateisystemen oder Teilen davon befasst. Das TAR-Format als Vertreter eines gängigen Archiv-Formats wurde ursprünglich für die Datenarchivierung auf Bändern konzipiert (Tape Archive). Es bietet zusätzlich zur Kapselung von Verzeichnisstrukturen, die Möglichkeit die Nutzdaten der Dateien komprimiert zu speichern. Die Kompression findet in dieser Arbeit jedoch keine Beachtung.

Auch in einer TAR-Datei sind neben den Nutzdaten der archivierten Dateien Metadaten zur Beschreibung der Datei enthalten. Die Kombinationen dieser Beiden wird Archive-Member genannt. Archive-Member werden durch zwei 512 Byte Blöcke, gefüllt mit Null-Bytes von einander getrennt. Da TAR-Dateien ursprünglich zur Archivierung auf Block-Speichermedien entwickelt wurden, sind diese Dateien in Blöcke aufgeteilt. Somit besteht eine TAR-Datei aus einer Aneinanderreihung von Archive-Membern durch Null-Byte-Blöcke von einander getrennt. Der Header einer archivierten Datei belegt einen 512 Byte-Block und enthält den Namen, die Berechtigungen, und weitere bekannte und TAR-spezifische Metadaten zur Dateibeschreibung. Die folgende Abbildung illustriert den Aufbau eines TAR-Archivs [FSF07].

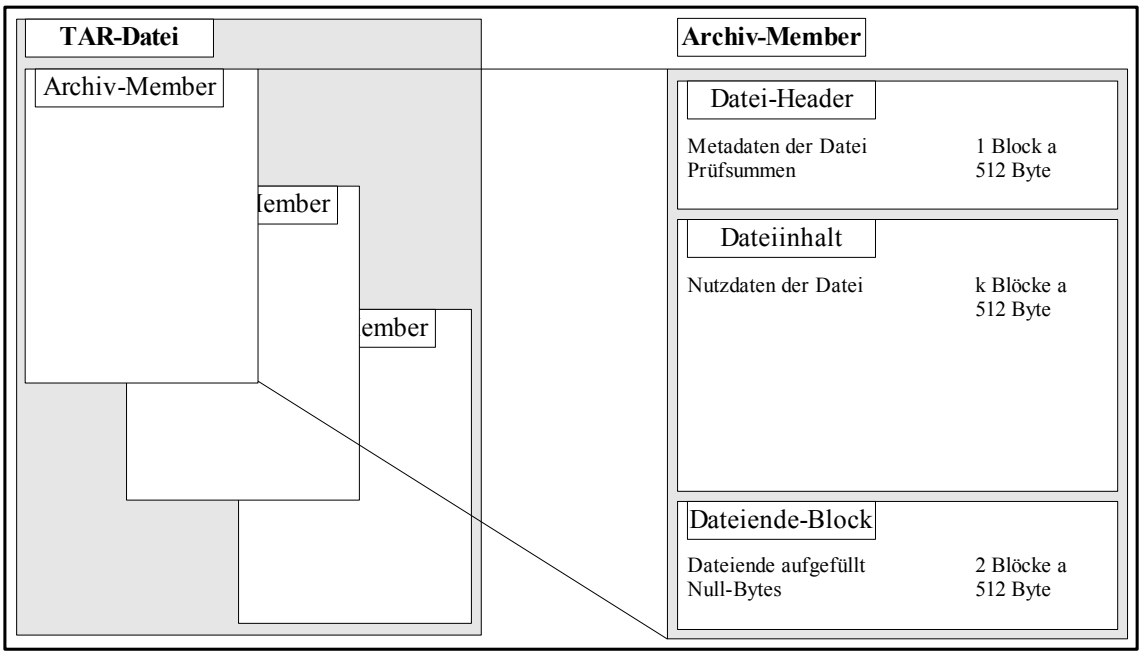


Abbildung 3.1: TAR-Archiv

Der Vorteil, den eine solche TAR-Datei bietet, ist die Möglichkeit die Datei ohne zusätzlichen Verwaltungsaufwand zu vergrößern. Ein wahlfreier Zugriff auf die archivierten Dateien ist jedoch, wegen des sequenziellen Aufbaus des TAR-Archivs nicht leistungsstark. Das Auflisten des TAR-Datei-Inhalts ist aus dem selben Grund ebenfalls verhältnismäßig langsam. Dazu kommt, dass das TAR-Archiv $3 \cdot 512\ Byte = 1{,}5\ KByte$ Zusatzlast je Datei produziert.

Das im nächsten Kapitel definierte Container-Archiv-Format wird, unter Berücksichtigung der oben aufgeführten Vorgaben und Einschränkungen, Ineffizienzen, wie sie bei der Verwendung der TAR-Formats ergeben, für den zugrunde liegenden Anwendungsfall des DKFZ minimieren. Dazu werden im Folgenden die Anzahl der Eingabe-/Ausgabe-Operationen und die Menge an zusätzlichen Informationen (Zusatzlast) betrachtet. Durch die Minimierung dieser beiden Faktoren kann eine Steigerung der Leistung erreicht werden, da vor allem die Eingabe-/Ausgabe-Operationen durch die vergleichsweise langsamen sekundären Datenträger wie Festplatten sehr zeit-intensiv sind.

3.3 Definition des Containerformats

Nach den vorangegangenen Überlegungen werden hier die Entscheidung für ein Containerformat getroffen. Das Containerformat ist wie folgt definiert.

Die eigentliche Containerdatei ist dreigeteilt aufgebaut.

Der erste Teil enthält den Header, welcher wiederum die Anzahl, der im Container, enthaltenen Dateien, die Gesamtlänge der Dateinamen dieser Dateien und eine Referenz auf das Inhaltsverzeichnis enthält. Diese Daten sind nicht zwingend vor der Erstellung der Containerdatei zu ermitteln, da der Header eine statische Größe besitzt. Es ist daher eindeutig festgelegt, wo der zweite Teil der Containerdatei beginnt und damit die Möglichkeit gegeben, den Header auch nachträglich einzufügen.

Zu beachten ist jedoch, dass die Anzahl an Bytes, die für einen Offset reserviert werden, variieren kann. So ist es möglich, auf einem *32-Bit*-System Dateien mit einer Größe von maximal $2^{31}\ Byte$ zu erstellen. Daraus resultiert die Verwendung von *4 Byte* für einen Offset. Unterstützt ein ein solches System LFS (Large File Support), können durchaus Dateien bis zu einer Größe von $2^{63}\ Byte$ erstellt werden [Jae05]. Dies bedeutet aber auch, dass dann *8 Byte* für einen Offset reserviert werden.

Der zweite Teil der Containerdatei, im Folgenden Dateiblock genannt, enthält die eigentlichen Dateien. Die Dateien werden an dieser Stelle in der Containerdatei positioniert, da es im Weiteren auch möglich sein soll, aus einer Anwendung heraus eine Containerdatei zu erstellen. Daher muss die Möglichkeit bestehen Dateien in die Container Datei zu schreiben, bevor festgestellt werden kann wie viele Dateien es letztendlich sein werden. Da die Anzahl der Dateien während des Ablaufs einer solchen Anwendung nicht festgestellt werden kann, kann auch nicht festgelegt sein wie Groß das Inhaltsverzeichnis der Containerdatei sein wird. Auch wenn diese Art der Erstellung einer Containerdatei nicht Teil dieser Arbeit ist, wird hier berücksichtigt, dass zukünftige Weiterentwicklungen des Konzepts in

dieser Richtung sehr sinnvoll sind.

Der dritte Teil enthält das Inhaltsverzeichnis des Containers, im Folgenden Index genannt. Dieser Teil enthält die zur Dateiidentifikation und Lokalisation nötigen Informationen und kann durch den im Header abgelegten Offset erreicht werden. Für jede Datei im Container wird einen Block, im Folgenden Indexblock genannt, mit folgenden Informationen angelegt. Er enthält eine Referenz auf den Namen der Datei, die Größe der Datei und den Offset. Der Dateiname wird dabei zur Identifikation der Datei verwendet.

Da die Dateinamen keine feste Länge haben, werden sie separat abgelegt. Der Indexblock, mit den restlichen Dateiinformationen, enthält jeweils eine Referenz fester Größe auf den entsprechenden Dateinamen.

Der Offset, also die Angabe wo in der Containerdatei die eigentliche Datei beginnt, entspricht der Distanz vom Beginn der Containerdatei zum eigentlichen Dateibeginn. Zur Ermittlung dieses Offsets ist der Offset der vorhergehenden Datei als auch die Dateigröße dieser Datei nötig.

Zusätzlich wird noch definiert, dass die Blöcke nach Dateinamen aufsteigend sortiert sind. Dadurch wird zwar die Komplexität und damit der Zeitaufwand zur Erstellung einer solchen Containerdatei vergrößert, aber dieses Zugeständnis wird sich, wie später noch gezeigt wird, vor allem beim wahlfreien Zugriff auf die Dateien in der Containerdatei, bezahlt machen.

Der Effizienz wegen werden also so wenig wie möglich Informationen über alle Dateien in einer sinnvollen Sortierung abgelegt. Damit wird eine minimale Zusatzlast realisiert.

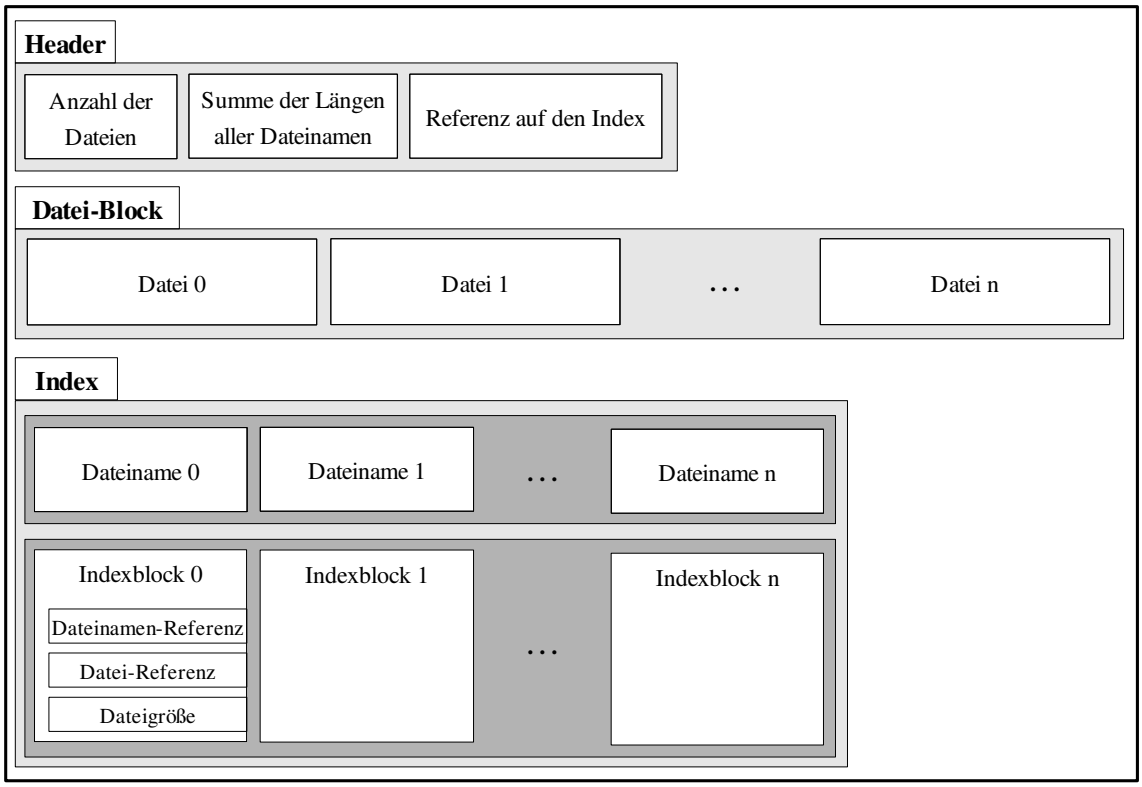


Abbildung 3.2: Containerdatei-Format

3.4 Definition/Herleitung resultierender Zusammenhänge

Dieses Unterkapitel dient dazu, einen Überblick über die verwendeten Datenfelder im Header und Inhaltsverzeichnis der Containerdatei zu ermöglichen. Dabei werden auch die Größen der Felder betrachtet, um im weiteren resultieren Zusammenhänge und Abhängigkeiten darzustellen. Die maximale Containerdateigröße ist eine der resultierenden Größen, genau wie die Zusatzlast pro enthaltener Datei. Zuvor werden die Datenfelder begrifflich definiert.

3.4.1 Begriffsdefinition

Die im Folgenden definierten Begriffe dienen einer besseren Übersichtlichkeit für spätere Betrachtungen. Die Namen selbst sind weitestgehend selbsterklärend und in ihrer Form so gewählt, dass sie der Variablen-Namen-Struktur einer Programmiersprache gerecht werden.

- Header

  numFiles - Anzahl aller, in der Containerdatei enthaltenen, Dateien

  sumNameLength - Summe der Längen aller Dateinamen in Byte

  indexOffset - Beginn des Inhaltsverzeichnisses relativ zum Beginn der Containerdatei in Byte

- Index

  fNameArray - Zeichenkette der Länge *sumNameLength*, die alle Dateinamen enthält

  indexArray - Array, welches der Dateianzahl (*numFiles*) entsprechend viele Indexblöcke enthält

- Indexblock

  fNameOffset - Beginn des Dateinamen im *fNameArray*

  fSize - Größe der Datei in Byte

  fOffset - Beginn der eigentlichen Datei relativ zum Beginn der Containerdatei in Byte

Es folgt daraus, dass die Größe des Headers konstant und die Größe der Indexblöcke konstant ist, da die Dateinamen in einem separaten Array gehalten werden. Damit wird erreicht, dass für die Dateinamen nicht mehr als der nötig Speicherplatz reserviert werden muss. Trotzdem bleibt die Möglichkeit bestehen Suchalgorithmen geringer Komplexität für den wahlfreien Zugriff einzusetzen, da weitgehend statische Strukturen betrachtet werden können.

3.4.2 Reservierte Größen in Byte

Dieses Unterkapitel gibt für die einzelen Datenfelder die verwendeten Datentypen und Größen in Byte wieder.

- *numFiles*

  4 Byte = 32 Bit werden für die Anzahl der in der Containerdatei enthaltenen Dateien reserviert. Auch wenn für den realen Anwendungsfall am DKFZ weniger Bit ausreichen würden, werden, um die Verwendungsmöglichkeiten möglichst nicht einzuschränken, bis zu 4 Milliarden Dateien vorgesehen.

- *sumNameLength*

  4 Byte = 32 Bit werden für die Summe der Längen der Dateinamen reserviert. Aufgrund der zuvor definierten Byte-Größe von *numFiles* wäre es theoretisch nötig, dass für *sumNameLength 40 Bit = 5 Byte* reserviert werden. Das folgt aus

  2^{32} *Dateinamen* $\cdot$ 2^{8} *Zeichen pro Dateiname* = 2^{40} *Zeichen*

  Es wird aber davon ausgegangen, dass die reservierten 4 Byte mehr als ausreichen. Als Begründung dienen folgendeAnnahmen:

  i) $numFiles \leq 2^{25} \approx 33{,}5 \cdot 10^{6}$

  ii) Länge des Dateinamen $\leq 2^{7} = 128$ *Zeichen*

  daher folgt

  2^{25} *Dateinamen* $\cdot$ 2^{7} *Zeichen pro Dateiname* = 2^{32} *Zeichen*

  dass 4 Byte genügen.

- *indexOffset*

  64 Bit = 8 Byte werden für den *indexOffset* reserviert. Das kann aber von System zu System variieren, da möglicherweise in einigen Umgebungen kein LFS (Large File Support) existiert. In solchen Fällen werden für Offsets nur *32 Bit = 4 Byte* reserviert. Daraus resultiert auf solchen Systemen eine maximale Containerdateigröße von 2 GByte.

- *Länge des Dateinamen*

  Da der Dateiname bei UNIX-Dateien nicht länger als 255 Zeichen sein darf, wird diese Länge auch für die Dateien in der Containerdatei festgelegt. Daher ergibt sich eine maximale Größe von 256 Byte für 255 Elemente des Dateinamen, da noch "\0" angefügt wird. Es werden jedoch nur soviel Byte reserviert wie tatsächlich für das Ablegen des Dateinamen nötig ist. Das heißt, für einen 10 Zeichen umfassenden Dateinamen werden 11 Byte reserviert.

- *fNameArray*

  Die Anzahl der Byte die für das *fNameArray* reserviert werden, liefert der Wert der Variable *sumNameLength*. Für dieses Array werden genau so viele Bytes reserviert, wie nötig sind, um eine Zeichenkette bestehend aus allen Dateinamen und den trennenden Null-Bytes zu halten.

- *fNameOffset*

  32 Bit = 4 Byte werden für den Offset des Dateinamen reserviert, da die Variable *sumNameLength* ebenfalls 4 Byte groß ist. Der Wert von *sumNameLength* beschränkt die Größe des *fNameArray* nach oben. Damit ist gewährleistet, dass *fNameOffset* auf jedes Byte im *fNameArray* verweisen kann.

- *fSize*

  32 Bit = 4 Byte werden für die Größe einer, im Container enthaltenen Datei reserviert. Damit lassen sich auch Container erstellen, die wenige aber bis zu 4 GByte große Dateien enthalten.

- *fOffset*

  64 Bit = 8 Byte werden auf Systemen, die LFS unterstützen und *32 Bit = 4* Byte auf Systemen ohne LFS-Unterstützung, für den Offset einer Datei reserviert.

- *indexArray*

  Die Größe des Speicherplatzes, der für das *indexArray* aufgewendet wird genügt folgendem Produkt

  $$sumFiles \cdot 16\ Byte,$$

  da für jede Datei im Container ein Indexblock mit folgenden Attributen im *indexArray* gespeichert wird.

  - *fNameOffset*, 4 Byte
  - *fSize*, 4 Byte
  - *fOffset*, 8 Byte

Mit den so definierten Größen wird im folgenden Abschnitt betrachtet, welche Auswirkungen für die Containerdatei bezüglich ihrer Gesamtgröße und der Pro-Datei-Zusatzlast auftreten.

3.4.3 Resultierende Abhängigkeiten (betrachtet mit LFS)

Die hier dargestellten Zusammenhänge setzen ein System voraus das Large File Support (LFS) unterstützt. Damit ist gemeint, dass das System mit Dateizeigern mit einer Mindestgröße von *64 Bit = 8 Byte* arbeitet. Auf Systemen, die mit *32 Bit = 4 Byte* großen Dateizeigern arbeiten verändern sich die Größen der hier mit Offset bezeichneten Datentypen. Damit kann auch die Gesamtgröße einer Containerdatei 2 GB nicht überschreiten.

Für die theoretische maximale Containerdateigröße *cFileSize* gilt,

$$\begin{aligned} cFileSize &= hSize + iSize + fBlockSize \\ &\leq 16\ Byte + 68\ GByte + 16\ EByte \\ &\approx 16\ EByte, \end{aligned}$$

mit der konstanten Headergröße *hSize,* für die gilt

$$hSize = 16\ Byte,$$

der maximalen Indexgröße *iSize,* für die gilt,

$$\begin{aligned} iSize &= nArraySize + numFiles \cdot iBlockSize \\ &\leq 2^{32}\ Byte + 2^{32} \cdot 2^{4}\ Byte \\ &= 68\ GByte, \end{aligned}$$

wobei *iBlockSize* die konstante Größe eines Indexblocks, und *nArraySize* die maximale Größe des Dateinamen-Array ist, für die gilt,

$$iBlockSize = 16\ Byte$$

$$\begin{aligned} nArraySize &= sumNameLength\ Byte \\ &\leq 2^{32}\ Byte \\ &= 4\ GByte \end{aligned}$$

und mit der maximalen Dateiblockgröße *fBlockSize,* für die gilt,

$$\begin{aligned} fBlockSize &\leq numFiles \cdot fSize \leq 2^{32} \cdot 2^{32}\ Byte \\ &= 16\ EByte. \end{aligned}$$

Die ermittelte Containerdateigröße von rund 16 ExaByte, folgt aus der Verwendung der maximalen Werte, die die Variableninhalte annehmen können. Containerdateien dieser Größe zu erstellen ist jedoch nur theoretisch möglich, da auch mit LFS (also 8-Byte-Dateizeigern) nur Dateigrößen bis 8 EByte adressiert werden können.

Dieses Betrachtung dient im Weiteren dazu, die untere Schranke für die Zusatzlast zu bestimmen, die bei der Verwendung von Containerdateien im besten Fall entsteht. Zum Vergleich werden die Größen der Zusatzlast einer TAR-Datei und eines Ext3-Verzeichnisses betrachtet, wobei auch hierbei nur von den theoretisch erreichbaren Größen ausgegangen wird. Die hier definierte Containerdatei als auch die TAR-Datei speichern die Metadaten der archivierten Dateien im Container bzw. im Archiv. Das Dateisystem speichert die Metadaten der Dateien in den Verwaltungsstrukturen. Um Vergleichbarkeit bezüglich der Zusatzlast-Größen zu realisieren, werden hier die Größen für *Inodes* und *Dentrys* des Ext2/3 Dateisystem zugrunde gelegt. Daher ist ein *Inode* konstant 120 Byte und ein *Dentry*, bei konstanter Dateinamenlänge von 128 Zeichen,192 Byte groß.

- *Zusatzlast – Containerdatei*

  Für eine Containerdatei existiert ein *Dentry* und ein *Inode* im Dateisystem. Außerdem wird der Header der Containerdatei als Zusatzlast gespeichert. Des Weiteren werden

für jede Datei ein Indexblock und der Dateiname verwaltet. Daher gilt für die Größe der Zusatzlast *COverheadSize* einer Containerdatei

$$\begin{aligned} &COverheadSize \\ &= 120\ Byte + 192\ Byte + hSize + numFiles \cdot (iBlockSize + Länge\ des\ Dateinamen) \\ &= 120\ Byte + 192\ Byte + 16\ Byte + 2^{32} \cdot (16\ Byte + 128\ Byte) \\ &\approx 576\ GByte, \end{aligned}$$

bei 16 EByte Nutzdaten.

- *Zusatzlast – TAR-Archiv*

  Auch für ein TAR-Archiv existiert ein *Dentry* und ein *Inode* im Dateisystem. Zusätzlich werden für jede Datei ein 512 Byte Block für Metadaten und zwei 512 Byte Blöcke für das End-Of-Archiv gespeichert. Damit gilt für die Größe der Zusatzlast *TOverheadSize* einer TAR-Datei

$$\begin{aligned} &TOverheadSize \\ &= 120\ Byte + 192\ Byte + Dateianzahl \cdot 1536\ Byte \\ &= 120\ Byte + 192\ Byte + 2^{32} \cdot 1536\ Byte \\ &\approx 6\ TByte, \end{aligned}$$

  bei ebenfalls 16 EByte Nutzdaten in 2^{32} archivierten Dateien.

- *Zusatzlast – Dateisystem*

  Dagegen entsteht bei der Verwendung eines Verzeichnisses mit 2^{32} enthaltenen Dateien für jede Datei ein *Inode* und ein *Dentry* im Dateisystem zusätzlich zu dem *Inode* und *Dentry* des Verzeichnisses. Daher gilt für die Größe der Zusatzlast *VOverheadSize* eines Verzeichnisses im Ext2/3 Dateisystems

$$\begin{aligned} &VOverheadSize \\ &= 120\ Byte + 192\ Byte + Dateianzahl \cdot (120\ Byte + 192\ Byte) \\ &= 120\ Byte + 192\ Byte + 2^{32} \cdot (120\ Byte + 192\ Byte) \\ &\approx 1{,}2\ TByte, \end{aligned}$$

  bei den oben genannten Größen.

Zusammenfassend wird festgestellt, dass die Zusatzlast, die eine Containerdatei verursacht, weniger als die Hälfte der Zusatzlast beträgt, die ein vergleichbares Verzeichnis im Ext2/3-Filesystem verursacht. Weiterhin wird durch Verwendung einer Containerdatei sogar nur ein Zehntel der Zusatzlast produziert die ein vergleichbares TAR-Archiv benötigt.

Es ist somit ein Containerformat definiert welches, bezüglich der Speichereffizienz, Vorteile gegenüber gängigen Verfahren zur Verwaltung von Dateien bietet.

Zusätzlich ist durch die Sortierung des Indexes gewährleistet, dass sowohl der sequentielle als auch der wahlfreie Zugriff, auf im Container enthaltene Dateien, effizient umgesetzt werden kann, da neben dem *IndexArray* keine zusätzliche Datenstruktur, wie zum Beispiel eine Baumstruktur, gespeichert werden muss. Der wahlfreie Zugriff auf Dateien mit Hilfe des sortierten *IndexArray* ermöglicht Zugriffe mit einer Komplexität von O(log n) zum Beispiel durch die Verwendung der binären Suche.

Auch die Auflistung der Dateiname kann nun effizient umgesetzt werden, da für ein *ls* ohne weitere Optionen zum Beispiel nicht der gesamte Index gelesen werden muss, sondern nur das *fNameArray* in dem die Dateinamen abgelegt sind.

4 Schnittstellendefinition

In diesem Abschnitt werden die Schnittstellen den Containerzugriff definiert. Dabei werden zwei Abstraktionsebenen betrachtet. Einerseits soll der Umgang mit Containerdateien von der Kommandozeile unterstützt werden und andererseits soll auch für Anwendungen eine API zum Umgang mit Containerdateien zur Verfügung stehen.

4.1 Kommandozeilenorientierte Schnittstellen

Um einen möglichst intuitiven Umgang mit Containerdateien zu erreichen, orientieren sich die Schnittstellen für die Kommandozeile an bekannten Schnittstellen für die Verzeichnismanipulation. Verzeichnisse können erstellt, deren Inhalt aufgelistet und enthaltene Dateien kopiert werden. Auch wenn dies nur eine sehr eingeschränkten Liste der Möglichkeiten darstellt, werden die folgenden Schnittstellen auf diese drei Funktionalitäten beschränkt. In dieser Definition wird daher festgehalten, dass Container von der Kommandozeile aus erstellt werden können. Des Weiteren, dass der Inhalt einer Containerdatei aufgelistet werden kann und als Drittes wird es ermöglicht eine Datei aus dem Container heraus zu kopieren. Es werden daher drei Programme *ctmk*, *ctls*, *ctfcp* für die Kommandozeile implementiert. Dabei steht *ctmk* für Container-make, *ctls* für Container-ls und *ctfcp* für Container-file-copy.

Kommandozeilenprogramm	**Parameter**	**Kurzbeschreibung**
ctmk	<Quellverzeichnisname> [<Containerdateiname>]	Container-make erstellt eine Containerdatei mit dem Inhalt des Quellverzeichnisses.
ctls	[-l] <Containerdateiname>	Container-ls listet den Inhalt einer Containerdatei mit oder ohne Metadaten auf.
ctfcp	<Dateiname> <Containerdateiname> <Zieldateiname>	Container-file-copy erstellt eine Kopie einer im Container enthaltenen Datei.

Tabelle 4.1: Kommandozeilen-API

In den folgenden Unterkapiteln werden die Kommandozeilenprogramme genauer betrachten. Dabei wird zu jedem Programm auf Details der Implementierung eingegangen. Um die Übersichtlichkeit zu wahren, wird an dieser Stelle auf das Zitieren von Quelltext verzichtet und zur Erläuterung des Ablaufes aller Kommandozeilenprogramme Pseudocode angegeben.

4.1.1 Containererstellung

Für die Erstellung von Containerdateien wird das Kommandozeilenprogramm *ctmk* implementiert.

Syntax: *ctmk <Quellverzeichnisname> [<Containerdateiname>]*

ctmk verlangt als Übergabeparameter einen Quellverzeichnisname und optional den Namen, unter dem der Container angelegt werden soll. Wobei beide Parameter als vollständige oder relative Pfade angeben werden können. Als Ergebnis entsteht eine Datei, die auf *.ct* endet. Sie enthält alle regulären Dateien des Quellverzeichnisses, den Container-Header und ein Inhaltsverzeichnis. Wobei der Aufbau der Datei dem oben definierten Containerformat entspricht.

4.1.1.1 *ctmk*-Implementierung

Um eine Containerdatei anzulegen verarbeitet *ctmk* die folgenden Informationen. Der Quellverzeichnisname *dirName* gibt das Verzeichnis an, dessen Inhalt in einer Containerdatei archiviert werden soll. Diese Containerdatei wird mit dem Containerdateinamen *containerName* und der Endung *.ct* angelegt. Wenn dieser Parameter nicht übergeben wird, wird die Containerdatei mit dem Namen *dirName.ct* in dem Verzeichnis angelegt, in dem sich das Quellverzeichnis befindet. Um die Informationen für den Header der Containerdatei zu ermitteln wird die Anzahl der Dateien *numFiles*, die Summe der Längen aller Dateinamen *sumNameLength* und die Position des Inhaltsverzeichnisses *indexOffset* bestimmt. Abschließend wird für jede Datei die Dateigröße *fSize*, die Position in der Containerdatei *fOffset*, der Dateiname *fName* und die Position des Dateinamen *fNameOffset* ermittelt und verarbeitet.

dirName	Name des Quellverzeichnisses
containerName	Name der Containerdatei
numFiles	Anzahl der archivierten Dateien
sumNameLength	Summe der Längen aller Dateinamen
indexOffset	Position des Inhaltsverzeichnisses
fSize	Größe einer Datei
fOffset	Position einer Datei
fName	Name einer Datei
fNameOffset	Position eines Dateinamen

Tabelle 4.2: *ctmk*-Begriffsdefinitionen

Wie *ctmk* die Informationen verarbeitet zeigt die folgende Abbildung. Der Pseudocode dient dabei zur Erläuterung der Funktionsweise von *ctmk*.

```
ctmk <Quellverzeichnisname> [<Containerdateiname>]
(1)  dirName = Quellverzeichnisname
(2)  Wenn übergeben
       (2.1.1) containerName = Containerdateiname
     Sonst
       (2.2.1) containerName = dirName.ct
(3)  lies Namen der Verzeichniseinträge von dirName
(4)  sortiere Namen von dirName lexikographisch aufsteigend
(5)  erstelle leeren Container-Header header
(6)  erstelle leere Liste indexList
(7)  für jeden Namen von dirName
       (7.1)   wenn Name eine reguläre in dirName bezeichnet
                 (7.1.1) erstelle ein leeres Listenelement indexBlock
                 (7.1.2) füge Name fName dem indexBlock hinzu
                 (7.1.2) füge Größe fSize dem indexBlock hinzu
                 (7.1.3) füge indexBlock an die indexList an
                 (7.1.4) inkrementiere numFiles für header
                 (7.1.5) berechne sumNameLength für header
(8)  erstelle Containerdatei mit Namen containerName
(9)  schreibe Dateien aus dirName entsprechend der indexList nach containerName
(10) erstelle leeres Dateinamen-Array fNameArray
(11) erstelle leeres Array indexArray für die restlichen Dateiinformationen
(10) für jeden indexBlock der indexList
       (10.1)  füge fName in fNameArray ein
       (10.2)  füge fNameOffset in indexArray-Element ein
       (10.3)  füge fSize in indexArray-Element ein
       (10.4)  berechne fOffset
       (10.5)  füge fOffset in indexArray-Element ein
(11) berechne indexOffset
(12) füge indexOffset in header ein
(13) schreibe fNameArray, indexArray, header nach containerName
Ende ctmk
```

Pseudocode 4.1: *ctmk*

Aus der vorangegangen Abbildung folgt, dass *ctmk* zuerst eine Auflistung aller Verzeich-

nisinhalte des Quellverzeichnisses einliest. Da das Containerformat per Definition ausschließlich für reguläre Dateien Verwendung finden soll, ist an dieser Stelle im Programmablauf noch nicht absehbar, wie viele Dateien tatsächlich im Container enthalten sein werden. Daher werden die Metadaten der regulären Datei, nach vorhergehender Prüfung, erst in einer verketten Liste gehalten. Auch wenn das bedeutet, mehr Speicherplatz zu benötigen, bietet dieses Vorgehen die Möglichkeit die notwendigen Informationen, wie Anzahl der Dateien *numFiles* und die Summe der Längen der Dateien *sumNameLength*, zu bestimmen. Auf das Ablegen der Positionen der Dateien *fOffset* in der *indexList* wird an dieser Stelle zugunsten der Speichereffizienz verzichtet, da diese später noch berechnet werden können. Mit den so gewonnen Daten ist es nun möglich Speicherplatz für die statischen Arrays *fNameArray* und *indexArray* zu allokieren, da die Länge des *fNameArray* durch *sumNameLength*, und die Länge des *indexArray* durch *numFiles* bestimmt sind. Beide Arrays werden im Anschluss mit den Informationen der *indexList* initialisiert. Wobei zusätzlich benötigte Informationen wie der *fNameOffset*, also der Verweis eines *indexArray*-Elements auf den zugehörigen Dateinamen *fName* im *fNameArray*, als auch der *fOffset* für jede Datei, und der *indexOffset* berechnet und gespeichert werden. Nachdem der Container-Header, die Dateien, das Dateinamen-Array *fNameArray* und das *indexArray* in die Datei geschrieben sind, liegt eine Containerdatei nach dem in Kapitel 3 definierten Format vor.

4.1.2 Auflistung des Containerinhalts

Eines der Hauptziele dieser Arbeit ist es das Anzeigen des Inhalts zu beschleunigen. Für Verzeichnisse steht dafür das Kommandozeilenprogramm *ls* zur Verfügung. Für eine Containerdatei wird ein Programm *ctls* implementiert. Dieses Programm ermöglicht das Auflisten des Containerinhalts, wobei die Ausgabe mit der von *ls* identisch ist. Für *ctls* wird auch die von *ls* bekannte Option *-l* implementiert, um nicht nur die Dateiname ausgeben zu können, sondern auch die Metadaten der Dateien.

Syntax: *ctls [-l] <Containerdatei-Name>*

Ohne die Option *-l* gibt *ctls* einen Dateinamen pro Zeile aus. Bei gesetzter Option *-l* wird für jede Datei im Container ein String ausgegeben, der der Ausgabe von *ls -l* entspricht. Das heißt es werden neben den Dateinamen auch die Metadaten ausgegeben. Die Datei-Information, die laut Containerdefinition nicht gespeichert wurden, wie zum Beispiel die Dateizugriffberechtigungen werden durch Metainformationen der Containerdatei ersetzt.

4.1.2.1 *ctls*-Implementierung

Zur Auflistung des Inhalts einer Containerdatei ist deren Name *containerName* an *ctls* zu übergeben. Nach dem der Header eingelesen wurde, kann mit den vorhandenen Informationen *numFiles* und *sumNameLength* Speicherplatz für das *fNameArray* und das *in-*

dexArray allokiert werden. Mit Hilfe des ebenfalls im header abgelegten *indexOffset* können die Daten für die Arrays in der Containerdatei lokalisiert und gelesen werden. Damit stehen die Dateinamen *fName* und deren Größen *fSize* zu Ausgabe zur Verfügung. Es kann noch zusätzlich notwendig sein die Metadaten der Containerdatei selbst einzulesen. Dies ist dann der Fall, wenn die Option *-l* gesetzt ist und daher alle nicht gespeicherten Metadaten der Dateien durch Metadaten des Container ergänzt werden.

containerName	Name der Containerdatei
numFiles	Anzahl der archivierten Dateien
sumNameLength	Summe der Längen aller Dateinamen
indexOffset	Position des Inhaltsverzeichnisses
fNameArray	Array der Dateinamen
indexArray	Array der Datei-Metadaten
fName	Name einer Datei
fNameOffset	Position eines Dateinamen
fSize	Größe einer Datei
permissions	Zugriffsrechte der Containerdatei
st_nlink	Anzahl Links auf die Containerdatei
st_uid	ID des Containerdatei-Besitzers
st_gid	ID der Containerdatei-Gruppe
st_mtime	Zeit der Containerdatei-Manipulation

Tabelle 4.3: *ctls*-Begriffsdefinitionen

Der folgende Pseudocode illustriert den Programmablauf von *ctls*. Wenn neben dem Containerdateinamen *containerName* auch die Option *-l* gesetzt ist, gibt *ctls* für jede im Container enthaltene Datei auch die Metadaten aus. Dazu wird ein String erstellt der die Zugriffsrechte *permissions*, die Anzahl der Links *st_nlink*, den Besitzer *st_uid*, die Gruppe *st_gid*, die Dateigröße *fSize*, den Zeitstempel der letzten Manipulation *st_mtime* und den Dateinamen *fName* enthält. Da im Inhaltsverzeichnis der Containerdatei per Definition nicht alle Metadaten zu den enthaltenen Dateien gespeichert sind, werden diese durch Metadaten der Containerdatei ergänzt. Diese müssen, für eine beliebige Anzahl von Dateien, nur einmal vom Dateisystem angefordert und geliefert werden und stellen damit keinen Ineffizienz bezüglich der Leistung dar. Neben den Metadaten des Containers sind der Header, und das Inhaltsverzeichnis, bestehend aus dem Array der Dateinamen *fNameArray* und dem *indexArray* einzulesen. Mit den so aufbereiteten Daten lässt sich für jede Datei der Ausgabe-String zusammensetzen. Die daraus resultierende Ausgabe entspricht dann exakt der, die das bekannte Kommandozeilenprogramm *ls* mit gesetzter Option *-l* liefert.

Ist die Option *-l* nicht gesetzt, implementiert *ctls* zugunsten der Effizienz ein anderes Vorgehen. Die Ausgabe ohne *-l* orientiert sich wiederum an der Ausgabe von *ls*. In diesem

Fall werden nur die Dateinamen *fName* ausgegeben. Es besteht daher keine Notwendigkeit für das Einlesen des gesamten Inhaltsverzeichnisses des Containers. Auch müssen die Metadaten der Containerdatei selbst nicht eingelesen werden. Es folgt daher, dass nur der Header und das Dateinamen-Array *fNameArray* eingelesen werden müssen. Da die Dateinamen per Definition aufsteigend sortiert abgelegt sind, ist auch keine weitere Aufbereitung der Daten nötig, so dass damit die Ausgabe erfolgen kann.

ctls [-l] <Containerdateiname>

(1) *containerName* = *Containerdateiname*
(2) öffne Datei *containerName*
(3) erstelle leeren Container-Header *header*
(4) lies *header* aus Datei *containerName*
(5) wenn *-l* übergeben
  (5.1.1) erstelle leeres *fNameArray*, *indexArray*
  (5.1.2) lies *fNameArray*, *indexArray* aus Datei *containerName*
  (5.1.3) schließe Datei *containerName*
  (5.1.4) lies Metadaten-Struktur *cStat* aus Datei *containerName*
  (5.1.5) extrahiere *permissions*, *st_link*, *st_uid*, *st_gid*, *st_mtime* aus *cStat*
  (5.1.6) für jede enthaltene Datei (jedes Element von *indexArray*)
    (5.1.6.1) Ausgabe:
    permissions, *st_nlink*, *st_uid*, *st_gid*, *fSize*, *st_mtime*, *fName*
    (5.1.6.2) Ende *ctls*
  sonst
  (5.2.1) erstelle leeres *fNameArray*
  (5.2.2) lies *fNameArray* aus Datei *containerName*
  (5.2.3) schieße Datei *containerName*
  (5.2.4) für jede enthaltene Datei (jedes Dateinamen aus *fNameArray*)
    (5.2.4.1) Ausgabe:
    fName

Ende *ctls*

Pseudocode 4.2: *ctls*

4.1.3 Kopieren enthaltener Dateien

Für den Fall, dass einzelne Dateien anderweitig verwendet werden sollen, wird *ctfcp* implementiert. Dieses Kommandozeilenprogramm ermöglicht die Entnahme einer Kopie einer Datei aus dem Container. Damit muss nicht der gesamte Container kopiert oder verschoben werden, wenn eine geringe Anzahl der enthaltenen Dateien anderweitig verwen-

det werden soll.

Syntax: *ctfcp <Dateiname><Containerdateiname><Zieldateiname>*

ctfcp erwartet als Eingabeparameter den Namen der Datei, den Namen der Containerdatei und den Namen unter dem die Kopie erstellt werden soll. Wobei der letzte Eingabeparameter optional ist, da einfach der Dateiname in Verbindung mit dem aktuellen Verzeichnis verwendet werden kann. Denkbar ist auch eine Option *-all* anzubieten, die eine Kopie aller Dateien erstellt.

4.1.3.1 *ctfcp*-Implementierung

ctfcp öffnet die Containerdatei *containerName* und liest den Header ein, um mit den enthaltenen Informationen *numFiles* und *sumNameLength* Speicher für das Inhaltsverzeichnis in Form der beiden Arrays *fNameArray* und *indexArray* zu reservieren. Nachdem, mit Hilfe des *indexOffset* beide Arrays mit Daten aus der Containerdatei *containerName* beliefert sind, wird die Datei mit Namen *fName* im Dateinamen-Array *fNameArray* des Containers gesucht. Im Falle der Existenz kann davon ausgehend die Position *fOffset* der Datei *fName* ermittelt werden. Abschießend wird eine Kopie der lokalisierten Datei *locatedFile* unter dem Namen *targetFile* erstellt.

containerName	Name der Containerdatei
fName	Name der gesuchten Datei
targetFile	Name, unter dem die Kopie abgelegt wird
locatedFile	Metadaten der gesuchten Datei
indexOffset	Position des Inhaltsverzeichnisses
numFiles	Anzahl der archivierten Dateien
sumNameLength	Summe der Längen alles Dateinamen
fNameArray	Array der Dateinamen
indexArray	Array der Datei-Metadaten
fNameOffset	Position eines Dateinamen
fOffset	Position der Dateien im Container
fSize	Größe einer Datei

Tabelle 4.4: *ctfcp*-Begriffsdefinitionen

Die Effizienz der Implementierung von *ctfcp* ist in großem Maße abhängig von der Effizienz der Suche die zur Lokalisation der Datei *fName* verwendet wird. Als Suchalgorithmus kommt in dieser Umsetzung die binäre Suche zur Anwendung. Dieser "Divide and Conquer"-Algorithmus hat seinen großen Vorteil in der geringen Komplexität von O(log n).

Als Nachteil besteht die Notwendigkeit, dass der Suchraum sortiert vorliegen muss. Daher wurde schon in der Definition des Containerformats festgelegt, dass das Inhaltsverzeichnis einer Containerdatei aufsteigend nach Dateinamen sortiert abgelegt ist. Der Zeitaufwand bei der Erstellung einer Containerdatei wird dadurch vergrößert. Da dies jedoch ein einmaliger Aufwand ist, im Gegensatz zur vielfachen Notwendigkeit eine Datei im Container zu lokalisieren, hat diese Umsetzung ihre Berechtigung, da ihre Vorteile überwiegen. Es kann daher mit sehr geringer Komplexität, und daraus folgend hoher Effizienz nach der zu kopierenden Datei *fName* gesucht werden.

```
ctfcp <Dateiname> <Containerdateiname> <Zieldateiname>
(1)   containerName = Containerdateiname
(2)   fName = Dateiname
(3)   Wenn kein Zieldateiname angegeben
        (3.1.1)  targetFile = fName
        sonst
        (3.2.1)  targetFile =Zieldateiname
(4)   öffne Datei containerName
(5)   erstelle leeren header
(6)   lies header aus Datei containerName
(7)   erstelle leeres  fNameArray, indexArray
(8)   lies fNameArray, indexArray aus Datei containerName
(9)   Binäre Suche nach fName auf fNameArray
(10)  Wenn fName im fNameArray existiert
        (10.1.1) liefere Referenz locatedFile auf die Datei
        sonst
        (10.1.2) Ende ctfcp
(11)  erstelle Datei targetFile
(12)  lese Datei locatedFile aus Datei containerName
(13)  schreib Datei locatedFile nach Datei targetFile
(14)  schließe Datei targetFile
(15)  schließe Datei containerName
Ende ctfcp
```

Pseudocode 4.3: *ctfcp*

Für Anwendungen, welche einen Container als effizientes Dateiarchiv verwenden werden, ist ebenfalls die Effizienz des Suchalgorithmus von großer Relevanz, da jeder lesende Zugriff auf eine Datei im Container, eine vorhergehende Lokalisation bedingt. Mit den Schnittstellen, die für Anwendungen zur Verfügung stehen, befasst sich das folgende Unterkapitel.

4.2 Definition der Anwendungs-API

Aus Sicht einer Anwendung, die statt der Verwendung standardisierter Dateisystem-Schnittstellen, als Datenquellen eine Containerdatei verwendet, werden in diesem Unterkapitel Schnittstellen für den Zugriff definiert. Dazu gehören das Lesen des Containerdateiinhalts, das Öffnen und Lesen einer im Container enthaltenen Datei und das Positionieren des Datei-Lese-Zeigers. Die folgenden Prozedur-Prototypen definieren eine API für diese Zugriffe.

Prozedur	**Parameter**	**Kurzbeschreibung**
ct_open	*containerName*	öffnet die Containerdatei und liest deren Header und das Inhaltsverzeichnis aus und liefert eine Struktur *containerDescriptor* mit Verweisen darauf
ct_readdir	*containerDescriptor* *prevName*	liest den Dateiname *prevName* und liefert den darauf folgenden Dateiname aus dem Inhaltsverzeichnis erlaubt damit iterativen Zugriff alle Dateinamen
ct_file_locate	*containerDescriptor* *fName*	lokalisiert die Datei mit übergebenen Namen *fName* und liefert eine Referenz auf deren Index-Eintrag
ct_file_open	*containerDescriptor* *locatedFile*	liefert eine Struktur vom Typ *cFile*, welche die geöffnete Datei repräsentiert
ct_file_size	*cFile*	liefert die Größe einer enthaltenen Datei
ct_file_read	*cFile* *buffer* *nbyte*	liest *nbyte* Byte aus der, durch *cFile* repräsentierten Datei in den *buffer*
ct_file_lseek	*cFile* *offset* *whence*	positioniert den Lese-Zeiger innerhalb der Datei *cFile* zum *offset*, welcher von *whence* aus bestimmt wird
ct_file_close	*cFile*	gibt den für *cFile* reservierten Speicher frei
ct_close	*containerDescriptor*	schließt eine Containerdatei

Tabelle 4.5: Anwendungs-API-Übersicht

Im Folgenden werden die Typ-Deklarationen und Prototypen der API ausführlicher be-

trachten. Die API ist in der Form einer C-Bibliothek mit Name *libct.c* implementiert. Die Deklarationen, die in diesem Kapitel erläutert werden, sind in der zugehörigen Header-Datei *libct.h* aufgeführt und können so ohne großen Aufwand in aufrufende Anwendungen eingebunden werden. Der zugehörige Quelltext ist im Anhang der Arbeit angefügt.

Entsprechend der in Kapitel 3 definierten Größen, werden zum Beispiel für die Anzahl der Dateien (*numFiles*) 32 Bit benötigt. Die Definition legt fest, dass die vollen 32 Bit zur Darstellung positiver Zahlen verfügbar sein müssen. Daher wird für *numFiles* und alle anderen Variablen, die der selben Definition unterliegen der Datentyp

typedef unsigned long int ***ulong32***

definiert. Der Header einer Containerdatei, mit seinen Attributen *numFiles*, *sumNameLength* und *indexOffset*, ist als Struktur

struct ***header*** *{ ulong32 numFiles*

ulong32 sumNameLength

off_t indexOffset }

definiert. Bei der Verwendung einer Containerdatei aus einer Anwendung heraus ist der Header ein zentraler Bestandteil. Mit den Daten, welche die Attribute liefern, lässt sich komfortabel bestimmen, wo das Inhaltsverzeichnis in der Containerdatei positioniert ist und wie viel Speicherplatz dafür benötigt wird. Das Einlesen des Headers bedarf dabei keiner zusätzlichen Informationen, da per Definition sowohl die Position, als auch die Größe des Headers feststehen. Das Inhaltsverzeichnis einer Containerdatei besteht aus zwei Arrays. Die *fNameArray* genannte Zeichenkette enthält alle Dateinamen der enthaltenen Dateien. Als Trennzeichen zwischen den Dateinamen wird eine binäre Null verwendet, da so jeder Dateiname als eigene Zeichenkette interpretiert werden kann. Die Anzahl der Elemente dieser Zeichenkette ist durch das Header-Attribut *sumNameLength* bestimmt. Das indexArray enthält für jede Datei eine Struktur

struct ***indexArray*** *{ ulong32 fNameOffset*

ulong32 fSize

off_t fOffset }.

Damit nimmt dieses Array alle verbleibenden Metadaten der Dateien auf. Dazu gehören der Verweis auf den Dateinamen *fNameOffset*, die Größe der Datei *fSize* und die Position *fOffset* der Datei innerhalb der Containerdatei. Das Header-Attribut *numFiles* bestimmt für dieses Array die Elementanzahl. Die Struktur

struct ***containerDescriptor*** *{ int cFD*

struct header header*

char fNameArray*

struct indexArray indexArray}*

repräsentiert eine geöffnete Containerdatei. Sie enthält, neben einem Dateizeiger *cFD*, Verweise auf die Strukturen *header*, *fNameArray* und *indexArray*. *containerDescriptor* ist damit die zentrale Struktur zur Verwendung einer Containerdatei, da sie den Zugang zu

den Verwaltungsstrukturen des Containerinhalts darstellt. Um eine enthaltene Datei als geöffnet zu repräsentieren ist die Struktur

*struct **cFile**{ struct containerDescriptor* containerDescriptor*

struct indexArray fData*

off_t curPosition }

implementiert. Das Attribut *fData* verweist auf das assoziierte Element im Inhaltsverzeichnis des Containers. Zusätzlich enthält diese Struktur einen Verweis auf den assoziierten *containerDescriptor* und die aktuelle Position *curPosition* des Lese-Zeigers der Datei. Durch diese Positionsangabe kann im Weiteren sichergestellt werden, dass es nicht zu Überschreitungen der Dateigrenzen kommen kann. Diese Gefahr besteht beispielsweise beim Lesen einer Datei im Container, da dem Dateisystem nur das Ende der Containerdatei bekannt ist, nicht jedoch das Ende einer enthaltenen Datei.

Die Verwendung einer Containerdatei bedingt das Einlesen aller zur Verwaltung des Inhalts notwendigen Daten. Diese Aufgabe, als auch das tatsächliche öffnen der Containerdatei ist in der Prozedur

struct containerDescriptor **ct_open**(containerName)*

implementiert. Prozedurintern wird die Containerdatei für den lesenden Zugriff geöffnet und deren Dateizeiger in der Struktur *containerDescriptor* abgelegt. Auch der Header und das Inhaltsverzeichnis wird in die entsprechenden Strukturen *header*, *fNameArray* und *indexArray* eingelesen und Verweise darauf in der *containerDescriptor*-Struktur abgelegt. Den Name der nächsten Datei im Container liefert die Prozedur

char **ct_readdir**(struct containerDescriptor* containerDescriptor,*

char prevEntry).*

Als Parameter sind der *containerDescriptor* und der Name der vorhergehenden Datei *prevName* zu übergeben. Den Name der ersten Datei innerhalb der Containerdatei liefert *ct_readdir*, wenn für *prevName* ein Null-Zeiger ist übergeben wird.

Das Öffnen einer enthaltenen Datei für den lesenden Zugriff setzt das vorhergehende Lokalisieren der Datei innerhalb des Containers voraus. Dafürsteht die Prozedur

struct indexArray **ct_file_locate**(struct containerDescriptor**

containerDescriptor, char fName)*

zur Verfügung. Sie verlangt die Struktur *containerDescriptor*, welche die beiden Arrays *fNameArray* und *indexArray* und die Anzahl der Dateien *numFiles* kapselt, als Parameter. Zusätzlich wird der Name der gesuchten Datei *fName* verlangt. Prozedurintern wird im Inhaltsverzeichnis mit dem Algorithmus "Binäre Suche" die Datei identifiziert. Als Rückgabe liefert *ct_file_locate* einen Zeiger auf das entsprechenden *IndexArray*-Element. Um eine lokalisierte Datei zu öffnen, ist die Prozedur

struct cFile **ct_file_open**(struct containerDescriptor* containerDescriptor,*

struct indexArray locatedFile)*

implementiert. Sie erstellt eine Struktur vom Typ *cFile* und liefert einen Verweis darauf

zurück. Als Parameter verlangt sie eine Struktur *containerDescriptor*, die eine Containerdatei repräsentiert und einen Verweis auf das *indexArray*-Element der zu öffnenden Datei. Diese Struktur bildet die Grundlage für das Lesen und das Positionieren des Lese-Zeigers einer enthaltenen Datei. Da der Container definitionsgemäß nur den lesenden Zugriff unterstützt, ist kein Parameter für die Art des Öffnens notwendig. Das Lesen des Inhalts einer enthaltenen Datei ermöglicht die Implementierung der Prozedur

size_t ***ct_file_read****(struct cFile* cFile, void* buffer, size_t nbyte).*

Syntaktisch orientiert sich dieser und die folgenden Prototypen an den bekannten Systemfunktionen *read*, *lseek* und *close*. Anstelle des Dateizeigers in der bekannten Liste der Parameter wird jedoch die Struktur *cFile* verlangt. Die Implementierung stellt für *ct_file_read* und *ct_file_lseek* sicher, dass der Lese-Zeiger der Datei, die durch *cFile* bestimmt ist, nicht über die Dateigrenzen hinaus positioniert werden kann. Als Rückgabe liefert *ct_file_read* die Anzahl der tatsächlich gelesenen Bytes. An dieser Stelle sei noch darauf hingewiesen, dass es sich bei der hier verwendete Implementierung um keine thread-safe Formulierung der Prozedur handelt. Das folgt aus der Notwendigkeit eines lseek-Systemaufrufs innerhalb der Implementierung. In zukünftigen Versionen der Prozedur *ct_file_read* ist eine threadsafe Umsetzung mit Hilfe von Mechanismen zum gegenseitigen Ausschluss denkbar.

Die Prozedur

ulong32 ***ct_file_size****(struct cFile* cFile)*

liefert die Größe einer enthaltenen Datei und verlangt als Parameter eine Struktur von Typ *cFile*.

off_t ***ct_file_lseek****(struct cFile* cFile, off_t offset, int whence)*

liefert die Position des Lese-Zeigers in Bytes vom Beginn der enthaltenen Datei. Als Parameter wird, neben einer *cFile*-Struktur auch der *offset* und *whence* verlangt. Der Parameter *whence* kann, wie der bekannten *lseek*-Systemfunktion die Werte *SEEK_SET* und *SEEK_CUR* annehmen. Wobei einerseits die Position des Lese-Zeigers vom Beginn, und andererseits von der aktuellen Position der Datei bestimmt wird.

Das schließen einer geöffneten Datei übernimmt die Prozedur

void ***ct_file_close****(struct cFile* cFile).*

Deren Hauptaufgabe ist das Freigeben des Speicherplatzes, der für die *cFile*-Struktur reserviert ist. Das schließen der Containerdatei selbst wird durch die Prozedur

int ***ct_close****(struct containerDescriptor* containerDescriptor)*

implementiert. Sie schließt die Containerdatei und gibt die reservierten Speicherbereich für die Arrays *fNameArray* und *indexArray*, für den Header und die Struktur *containerDescriptor* selbst frei.

Die Fehlerbehandlung resultiert aus den gekapselten Systemfunktionen. So ruf die Prozedur *ct_open* beim öffnen einer Containerdatei intern die bekannte Systemfunktion *open* auf. Diese wiederum setzt im Fehlerfall die Variable *errno*, welche im ISO C Standard definiert ist.

Zusammenfassend wird festgehalten, dass durch die so definierte API, Anwendungen eine Containerdatei als Datenquelle nutzen können. Die definierten Prototypen erlauben einen intuitiven Umgang mit Dateien, die Inhalt eines Container sind. Ermöglicht wird dies durch die Verwendeten Bezeichner und den Syntax, der sich an bekannten Systemfunktionen orientiert.

Die Effizienz der, in den vorhergehenden Kapiteln, festgelegten Definitionen, Umsetzungen und Implementierungen, wird im folgenden Kapitel 5 genauer betrachtet. Dabei wird durch Test verifiziert, dass die Verwendung von Containerdateien Leistungsvorteile bietet.

5 Evaluation

Zur Evaluierung der Leistung des Containerformats in Verbindung mit den, zur Manipulation implementierten, Kommandozeilenprogrammen und der definierten API werden Tests durchgeführt. Es dazu werden die Zeiten gemessen, die notwendig sind, um ein Verzeichnis, eine Containerdatei und TAR-Archiv zu erstellen. Im Anschluss werden die Zeiten für das Auflisten der jeweiligen Inhalte, das heißt die enthaltenen Dateien, gemessen und abschließend werden Zeiten für den lesenden Zugriff auf Datei-Inhalte ermittelt. Mit der Auswertung der Testdaten und detaillierten Beschreibungen der Testumgebung Testabläufe befassen sich die folgenden Unterkapitel.

5.1 Testumgebung

Als Testumgebung wird ein reservierter Cluster-Knoten der Arbeitsgruppe um Prof. Dr. Thomas Ludwig genutzt. Die Tests werden auf einer 35 GByte RAID-Partition durchgeführt, die in vorangegangenen Test als sehr leistungsstark auffiel. Weitere Hardware-Details sind im Folgenden aufgelistet.

- Zwei Intel Xeon 2GHz CPUs
- Intel Server Board SE7500CW2
- 1 GB DDR-RAM
- 80GB IDE HDD
- 450 Watt Single Power Supply
- Debian Sarge (Linux 2.6.19-5-pvs)
- RAID-Controller Promise FastTrack TX2300 ()
- RAID0 (Striping): Two 160 GB S-ATAII HDDs

Getestet wird einerseits auf Ext3 und anderseits auf ReiserFS als Dateisystem.

5.2 Test: Erstellung

Dieser Test dient dem Vergleich, inwiefern sich die Zeiten zur Erstellung eines Verzeichnisses, einer Containerdatei und eines TAR-Archivs, bei unterschiedlichen Dateianzahlen unterscheiden.

Für das Erstellen eines Verzeichnisses wurde für diesen Test ein Shell-Skript mit Namen

create_testfiles.sh implementiert. Dieses Skript verlangt als Parameter den Name, des zu erstellenden Verzeichnisses. Zusätzlich muss die Dateianzahl, und die Dateigröße angegeben werden. Mit den übergebenen Informationen erstellt *create_testfiles.sh* eine Datei der gewünschten Größe, deren Inhalt Zufallszahlen sind. Diese Datei wird im Anschluss entsprechend oft kopiert, so dass letztendlich ein Verzeichnis mit den gewünschten Eigenschaften zur Verfügung steht.

Syntax: *create_testfiles.sh <Verzeichnisname> <Dateianzahl> <Dateigröße>*

Die Namen der enthaltenen Test-Dateien sind willkürlich nach dem Muster *test_file_x* angegeben, wobei *x* einer Durchnummerierung entspricht. Zusätzlich gilt, dass x auf sechs stellen mit vorangestellten 0-en aufgefüllt ist, so dass die längen-lexikographische Sortierung der Dateinamen der numerischen Sortierung nach *x* entspricht.

Für das Erstellen einer Containerdatei kommt das im Kapitel 4 betrachtete Kommandozeilenprogramm *ctmk* zur Anwendung. Als Quellverzeichnis dient das durch *create_testfiles.sh* erstellte Verzeichnis. Dieses Verzeichnis ist ebenfalls das Quellverzeichnis für das zu erstellende TAR-Archiv, welches mit dem *tar*-Kommando erstellt wird.

Testfall	Dateianzahl	Dateisystem	Dateigröße
1	10	Ext3	4 KByte
2	100		
3	1.000		
4	10.000		
5	100.000		
6	1.000.000		
7	10	ReiserFS	
8	100		
9	1.000		
10	10.000		
11	100.000		
12	1.000.000		
13	10	Ext3	1024 KByte
14	100		
15	1.000		
16	10.000		
17	10	ReiserFS	
18	100		
19	1.000		
20	10.000		

Tabelle 5.1: Testfälle

Der Test deckt die in der Tabelle gezeigten Testfälle ab, wobei für jeden der 20 Fälle je-

weils ein Verzeichnis, eine Containerdatei und TAR-Archiv, mit der entsprechenden Dateianzahl, Dateigröße und auf dem entsprechenden Dateisystem, erstellt wird. Auf die vier Testfälle, welche 100.000 1.000.000 Dateien der Größe 1024 KByte auf beiden Dateisystemen erstellen, wurde verzichtet, da der zur Verfügung stehen Festplatten-Platz nicht ausgereicht hätte. Dies stellt aber keinen Mangel dar, da sich die Tendenzen auch mit den geringeren Dateianzahlen aufdecken lassen. Mit dem Ablauf und der Auswertung der Messdaten befassen sich die folgenden Unterkapitel.

5.2.1 Testablauf: Erstellung

Hauptziel dieses Test ist es, neben dem Sammeln aussagekräftiger Daten zur Evaluation der Leistungsfähigkeit des Containerformats, Reproduzierbarkeit der gemessen Werte zu gewährleisten. Daher ist der Testablauf so konzipiert, das Cache-Effekte weitergehend ausgeschlossen werden können.

Zu Beginn jedes Testfalls wird mit dem Kommando *mkfs* eines der beiden Dateisysteme auf einer Partition erzeugt. Da das Resultat dieses Kommandos ein leeres Dateisystem ist, sind an dieser Stelle im Test Cache-Effekte ausgeschlossen. Aufeinander folgend werden anschließend die Aufrufe *create_testfiles.sh*, *ctmk* und *tar -cvf* bearbeitet. Um auch hier faire Bedingungen bezüglich möglicher Cache-Effekte zu erreichen, wird nach dem Erstellen des Verzeichnisses, als auch nach dem Erstellen der Containerdatei mit den Kommandos *umount* und *mount* das Dateisystem aus dem Verzeichnisbaum aus- und wieder eingehängt. Das jeweiliger Erstellen wird durch ein vorangestelltes *time*-Kommando gekapselt, so dass die Ausführungszeiten in ein Testprotokoll umgeleitet werden können und damit zu Auswertung zur Verfügung stehen.

5.2.2 Auswertung: Erstellung

Dieses Unterkapitel stellt die Zeiten, die zur Erstellung eines Verzeichnisses, einer Containerdatei und eines TAR-Archivs gemessen wurden, einander gegenüber. Die 20 Testfälle zugrunde legend wurden vier Diagramme erstellt. Jedes fasst eine Teilmenge der Testfälle zusammen. Die vier folgenden Abbildungen zeigen, nach Dateisystemen und -größen getrennt, wie sich die Erstellungszeiten bei zunehmender Dateianzahl entwickeln. Die jeweilige y-Achse dient der Darstellung der Zeit in Millisekunden und die x-Achse der Anzahl der Dateien in Potenzen von 10. Damit wird untersucht, ob die Erstellung einer Containerdatei zeitliche Nachteile gegenüber der Erstellung eines Verzeichnisses im Dateisystem oder eines TAR-Archivs verursacht. Dieser Test ist vor allem interessant für eine mögliche Erweiterung der Prozedur-orientierten API dahingehend, direkt aus Anwendungen heraus Containerdateien zu erstellen.

Das erste Diagramm deckt die Testfälle 1-6 ab und ermöglicht damit den zeitlichen Vergleich von Containerdatei, Verzeichnis und TAR-Archiv bei einer Dateigröße von 4

KByte auf einem Ext3 Dateisystem.

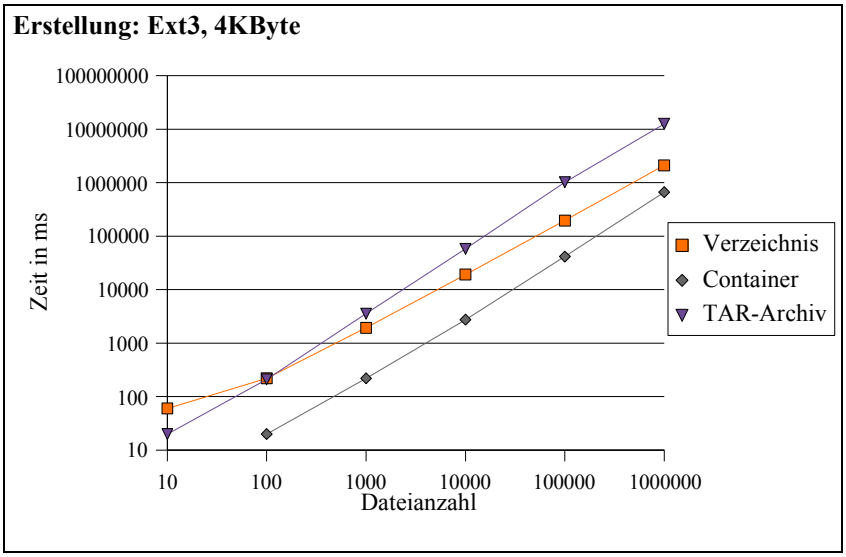


Diagramm 5.1: Erstellungszeiten Ext3, 4KByte

Das zweite Diagramm deckt die Testfälle 7-12 ab und unterscheidet sich dahingehend vom ersten, dass Containerdatei, TAR-Archiv und Verzeichnis auf dem Dateisystem ReiserFS erstellt werden. Auf beiden Dateisystem benötigt das Erstellen einer Containerdatei, bei gegebenem Quellverzeichnis deutlich weniger Zeit als das Erstellen eines Verzeichnisses mit der gleichen Menge enthaltener Dateien. Auf beiden Dateisystemen ist die

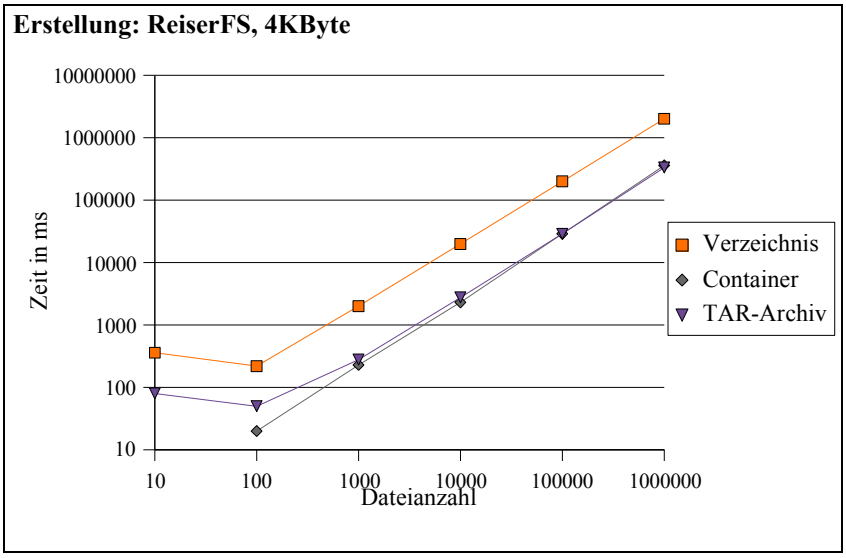


Diagramm 5.2: Erstellungszeiten ReiserFS, 4KByte

Containererstellung bei kleinen Dateianzahlen, um den Faktor 10 schneller als die Erstellung eines Verzeichnisses. Auch wenn der Leistungsvorteil bei zunehmender Dateianzahl abnimmt, ist jedoch auch bei einer Million Dateien noch mindestens ein Faktor 3 zugunsten der Containerdatei dokumentiert.

Die im folgenden betrachteten Diagramme zeigen die Zeiten die für die Erstellung von Containerdatei, Verzeichnis und TAR-Archiv, bei Dateianzahlen von 10 bis 10.000 und einer Dateigröße von 1024 KByte benötigt wurden. Damit decken sie die verbliebenen Testfälle 13-20 ab.

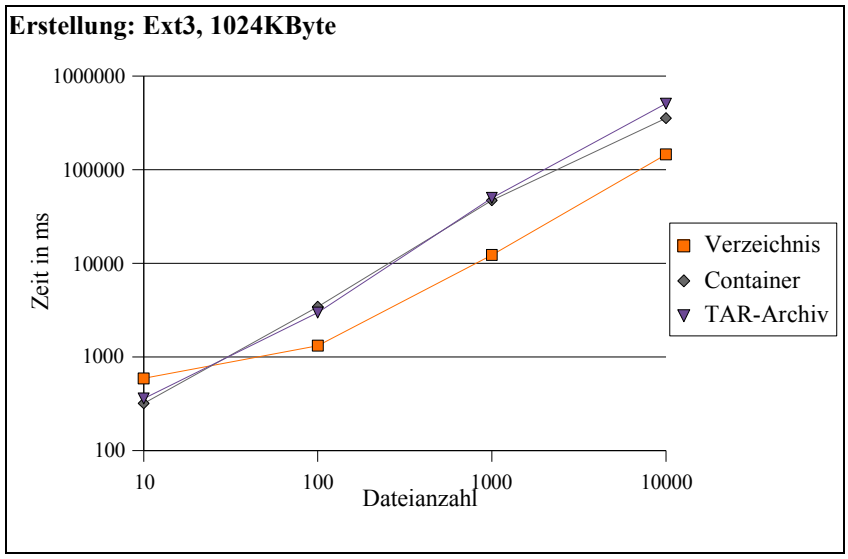


Diagramm 5.3: Erstellungszeiten Ext3, 1024KByte

Ab einer Dateianzahl von 100 Dateien kann hier die Erstellung eines Verzeichnisse um den Faktor 3 bessere Zeiten als für die Erstellung von Containerdatei und TAR-Archiv aufweisen. Dieses Verhalten lässt sich bei gleichen Voraussetzungen auch auf dem Dateisystem ReiserFS nachweisen, wie das folgende Diagramm belegt.

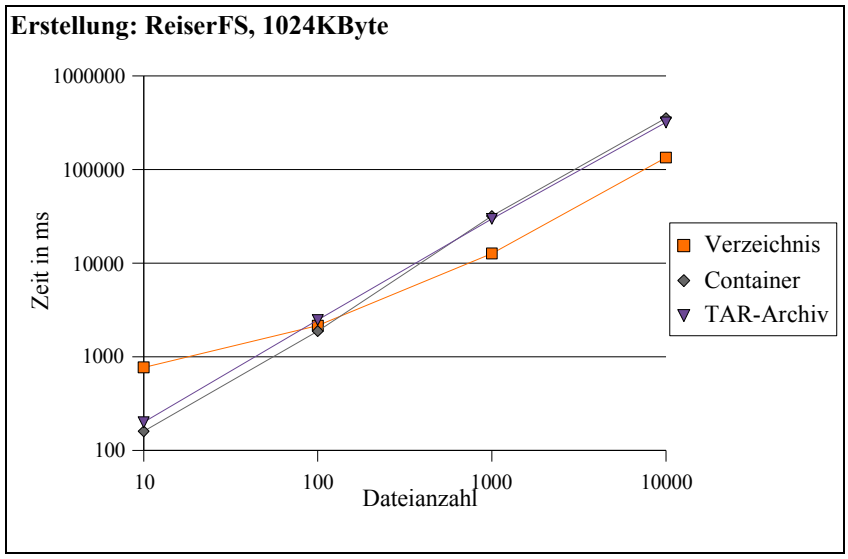


Diagramm 5.4: Erstellungszeiten ReiserFS, 1024KByte

Aufgrund der Tatsache, dass die Erstellung der Containerdatei und des TAR-Archivs gleichermaßen mehr Zeit in Anspruch nehmen, muss die Ursache für diese gegensätzliche

Verhalten, zwischen kleinen und großen Dateien, im Dateisystem begründet liegen. Zur Erstellung einer Containerdatei als auch eines TAR-Archivs ist es in der derzeitigen Umsetzung notwendig, die Dateiinhalte als auch die Metadaten der Dateien, die archiviert werden sollen, vom Dateisystem anzufordern. Da in diesem Testablauf großer Wert auf die Reproduzierbarkeit gelegt wird, werden die Kommandos zur Erstellung bei leeren Caches aufgerufen. Das heißt, dass für Zugriffe auf Dateiinhalte als auch die Metadaten der Dateien, ein Vielzahl von zeit intensiven Festplattenzugriffen nötig sind. Zudem müssen bei der Erstellung der Containerdatei und des TAR-Archivs die Dateien einmal aus dem Quellverzeichnis gelesen und einmal ins Zielarchiv geschrieben werden. Bei der Erstellung eines Verzeichnisses wird hingegen nur eine Datei, welche nach der Erstellung im Cache vorliegt, wieder und wieder geschrieben. Daher ist es nicht überraschend, dass die Erstellung eines Verzeichnisses bei großen Dateien zeitlich im Vorteil ist. Es ist vielmehr überraschend, dass das Erstellen eines Containers, trotz der geraden beschriebenen Problematik bei kleinen Dateien bis zu einer ganzen Größenordnung weniger Zeit in Anspruch nimmt.

Als Fazit wird festgehalten, dass in der jetzigen Umsetzung die Erstellung einer Containerdatei bei kleinen Dateien einen wesentlichen Vorteil bezüglich der benötigten Zeit gegenüber der Erstellung eines TAR-Archivs oder eines Dateisystem-Verzeichnisses hat. Außerdem ist auch bei großen Dateien ein zeitlicher Vorteil für die Containererstellung zu erwarten, wenn eine spätere Umsetzung nicht auf ein Quellverzeichnis und damit Lese- und Schreibzugriffe, sondern auf einer direkten Erstellung aus Anwendungen heraus mit ausschließlichen Schreibzugriffen basiert.

5.3 Test: Auflistung des Inhalts

Dieser Test beschreibt eines der Hauptziele dieser Arbeit. Es wird damit die Effizienz der Auflistung des Inhalts eines Verzeichnisses, einer Containerdatei und eines TAR-Archivs getestet. Dabei werden einerseits die enthaltenen Dateinamen und andererseits zusätzlich zu den Dateinamen noch die Metadaten der Dateien aufgelistet. Für ein Verzeichnis kommen dafür die bekannten Kommandos *ls* und *ls -l* zur Anwendung, wobei die Option *-l* neben den Dateinamen, also den Verzeichniseinträgen die Metadaten liefert. Der Inhalt des TAR-Archivs wird analog mit den Kommandos *tar -tf* und *tar -fvf* abgefragt. Für die Containerdatei wird das in Kapitel 4 betrachtete Kommandozeilenprogramm *ctls* zur Auflistung der enthaltenen Dateien verwendet und für die Ausgabe mit Metadaten um die Option *-l* ergänzt. Diesem Test liegen die 20 Testfälle des Erstellungstests zugrunde.

5.3.1 Testablauf: Auflistung des Inhalts

Der Ablauf dieses Tests baut auf dem vorhergehenden Test auf. Nachdem aus der Bearbei-

tung eines Testfalls des Erstellungstests die Existenz eines Verzeichnisses, einer Containerdatei und eines TAR-Archivs mit identischem Inhalt resultiert, wird die Effizienz der Auflistung der Inhalte getestet. Dazu implementiert der Auflistungstest folgendes Vorgehen. Zunächst werden für Verzeichnis, Container, und TAR-Archiv jeweils zehn Messungen durchgeführt. Dabei wird mit Hilfe des *time*-Kommandos die Zeit gemessen die zur Auflistung der Namen der enthaltenen Dateien nötig ist. Fünf dieser Messung werden mit leerem (cold) Cache und fünf mit gefülltem (hot) Cache durchgeführt. Im Anschluss daran wird durch identisches Vorgehen das Auflisten der Dateinamen und zugehörigen Metadaten gemessen. Das leeren des Caches geschieht durch das Aus- und Einhängen des Dateisystems mit den Kommandos *umount* und *mount*. Das fünffache Wiederholung der Messungen lässt eine Auswertung mit Mittelwerten zu.

5.3.2 Auswertung: Auflistung des Inhalts

Das erste Diagramm stellt die gemessen Zeiten für die Testfälle 1-6 mit leerem Cache ohne Metadaten dar.

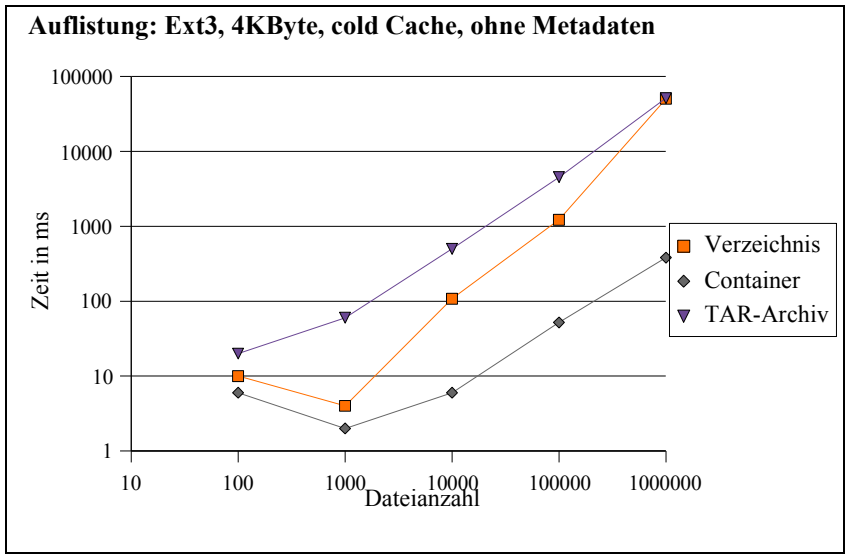


Diagramm 5.5: Auflistung des Inhalts Ext3, 4KByte, cold Cache, ohne Metadaten

Deutlich ersichtlich wird, dass das Auflisten der Namen der enthaltenen Dateien einer Containerdatei mit Abstand effizienter ist als bei den beiden Kontrahenten. Die Zeiten für das Listen des TAR-Archivinhalts, leiden unter dessen sequentiellem Aufbau, da dieser ein vollständiges Durchlaufen des gesamten Archivs bedingt.

Für Verzeichnisse zeichnet sich die Tendenz ab, dass je größer die Dateianzahl ist, desto mehr Zeitaufwand ist notwendig, um den Namen einer enthaltenen Datei aus dem *Dentry*-Objekt des Verzeichnisses zu extrahieren. Die Begründung dafür liegt in der notwendigen Indirektion im *Inode* des Verzeichnisses, bedingt durch die großer Anzahl von *Dentry*-Objekten für die enthaltenen Dateien. Diese Problematik hat zur Folge, dass, wie im Kapitel

2 erläutert, Caching-Strategien des Betriebssystemkerns an Effizienz verlieren, da das Lesen der entsprechenden Datenblöcke von der Festplatte, wegen mangelnder räumlicher Nähe auf dem Datenträger, ineffizienter wird. Als Folge dessen dauert das Auflisten des Verzeichnisinhalts bei einer einer Million Dateien so lange wie das Auflisten des Inhalts des sequentielle zu lesenden TAR-Archivs und um mehr als Faktor 100 länger als bei einer Containerdatei.

Das folgende Diagramm stellt die Ausführungszeiten dar, wenn zusätzlich zu den Dateinamen die Metadaten der enthaltenen Datei aufgelistet werden.

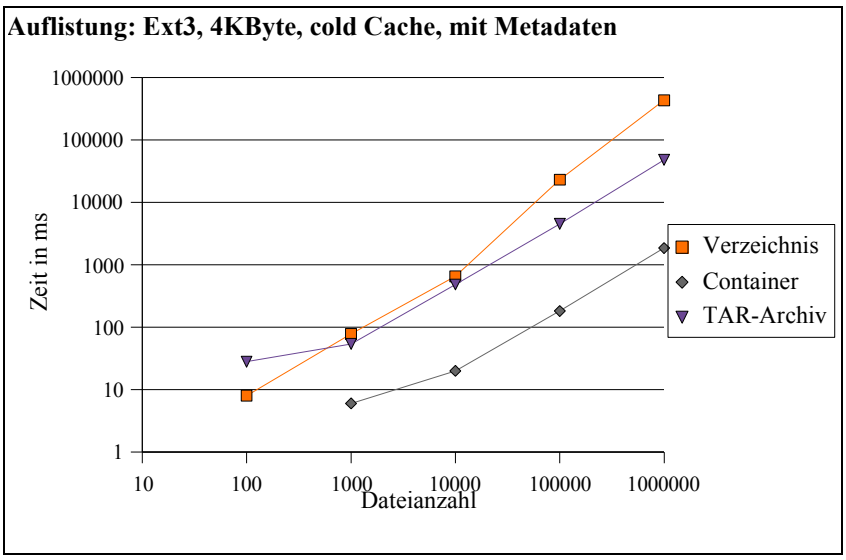


Diagramm 5.6: Auflistung der Inhalte Ext3, 4KByte, cold Cache, mit Metadaten

Während die Ausführungszeiten für das TAR-Archiv nahezu identisch bleiben, da wieder ein vollständiges Durchlaufen des Archivs notwendig ist, nehmen die Zeiten für das Auflisten des Containerinhalts um den Faktor 5 zu. Die Zeit für das Auflisten des Verzeichnisinhalts verzehnfacht sich sogar. Wie beim Test ohne Metadaten sind auch hier die selben Tendenzen abzulesen. Die zusätzlich benötigte Zeit bei einer Containerdatei resultiert daher, dass nun nicht nur das Dateinamen-Array *fNameArray* ausgewertet werden muss, sondern auch das *IndexArray* welches die Metadaten der Dateien liefert. Beim Verzeichnis ist der Grund für die verlangsamte Ausführung, dass neben dem Lesen der *Dentry*-Objekte, um die Dateinamen zu erhalten, auch die zugehörigen *Inodes* jeder einzelnen Datei lokalisiert und gelesen werden müssen. Das führt dazu, dass das Auflisten des Inhalts einer Containerdatei mit Metadaten, bei einer Million enthaltenen Dateien nur 20 mal schneller als das TAR-Archiv, aber um den Faktor 200 schneller als das entsprechende Verzeichnis ist.

Die vorherigen Test sind mit geleertem Cache ausgeführt worden. Da die Größenordnungen, mit denen die drei Kontrahenten vom Cache profitieren, sowohl für den Test mit als auch ohne Metadaten nahezu identisch sind, wird hier nur auf den Fall mit Metadaten eingegangen. Zur Visualisierung dient das nachfolgende Diagramm.

Bis zu einer Dateianzahl von 100.000 profitiert das TAR-Archiv dergestalt, dass sich Zei-

ten für das Auflisten mindestens halbieren. Bei einer Million Dateien jedoch kann keine Verringerung der Zeit gegenüber leerem Cache mehr nachgewiesen werden. Die Ursache dafür ist, dass für das Cachen des 4 GByte großen Archivs nicht genügend Hauptspeicher zur Verfügung steht. Da für das Cachen der Metadaten des Verzeichnisinhalts weniger als 800 MByte benötigt werden, die in den Hauptspeicher passen, wird für das Auflisten mit gefülltem Cache 50% weniger Zeit als mit leerem Cache benötigt. Das Auflisten des Containerinhalts benötigt nahezu die gleiche Zeit, wie ohne Cache. Sie verbessert sich nur um weniger als 5%. Trotzdem ist das Auflisten des Inhalts einer Containerdatei mit einer Million Dateien mehr als 25 mal schneller als das eines TAR-Archivs und mehr als 90 mal schneller als das eines Verzeichnisses.

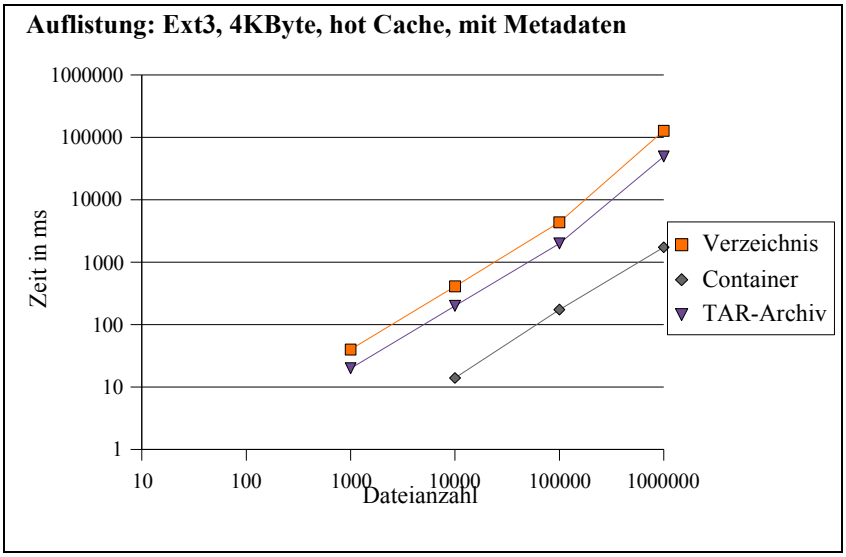


Diagramm 5.7: Auflistung der Inhalte Ext3, 4KByte, hot Cache, mit Metadaten

Das Ausführen der oben betrachteten Tests mit 1024 KByte großen Dateien führt zu identischen Schlussfolgerungen und wird daher nur kurz betrachtet. Da die Tests nur mit 10 bis 10.000 Datei zur Ausführung gebracht werden konnten beschränkt sich der Wissensgewinn sich auf die Folgenden Fakten. Das Auflisten des Verzeichnisinhalts verlängert sich auf das bis zu siebenfache bei leerem Cache. Die Ursache dafür liegt in der noch räumlich noch größeren Distanz der benötigten Daten auf dem Datenträger. Indiz dafür ist, dass sich bei gefülltem Cache die Zeiten für das Auflisten nicht von denen der 4 KByte Dateien unterscheiden. Für Containerdatei sind die benötigten Zeiten nahezu identisch, da die Dateigrößen keinen Einfluss auf die Größe der zu lesenden Daten des Inhaltsverzeichnisses haben. Für das Listen des TAR-Archivinhalts wird bis zu 200 mal die Zeit benötigt, die mit den 4 KByte Dateien nötig war. Da die Dateien mit 1024 KByte auch 256 mal so groß wie die 4 KByte Dateien sind, liegt auch hier wieder die Ursache in der Notwendigkeit das TAR-Archiv vollständig durchlaufen zu müssen, um dessen Inhalt aufzulisten.

Um Vollständigkeit zu gewährleisten sind alle, an dieser Stelle nicht aufgeführten Diagramme sowie die zugehörigen Testprotokolle im Anhang aufbereitetet. Dies gilt auch für die folgende Betrachtung der Test auf dem Dateisystem ReiserFS.

Auf dem Dateisystem ReiserFS wurden die selben Tests ausgeführt wie auf Ext3. Die folgenden Diagramme illustrieren die gemessenen Zeiten. Ein Unterschied zwischen Ext3 und ReiserFS besteht darin, dass ReiserFS kleine Datei anders verwaltet als große. Ext3 hingegen macht in dieser Beziehung keine Unterschiede. ReiserFS speichert kleine Dateien direkt in den *Inodes* der Verzeichnisse zu denen sie gehören. Damit kann der zur Verfügung stehende Festplattenplatz effektiver genutzt werden. Auch wird damit die Lokalität der Metadaten des Verzeichnisses und der enthaltenen Dateien und deren Nutzdaten für kleine Dateien höchstmöglich erhöht. Daraus resultiert, dass die Lese- und Schreibköpfe der Festplatte weniger große Wege zurücklegen müssen. Dadurch erreicht ReiserFS, besonders bei großen Dateianzahlen kleiner Dateien deutlich bessere Zeiten als Ext3. Die gemessen Zeiten für das Auflisten der Verzeichnisinhalte belegen die höhere Effizienz bei Metadatenoperationen.

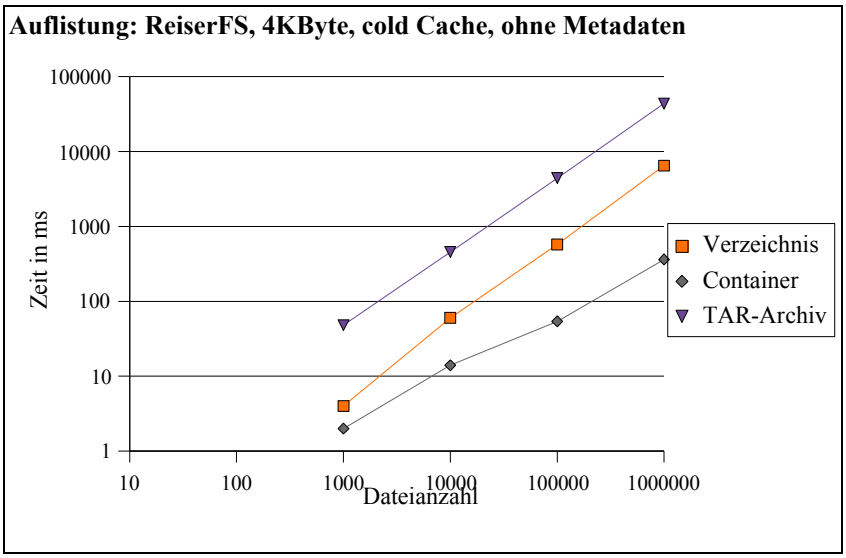


Diagramm 5.8: Auflistung des Inhalts ReiserFS, 4KByte, cold Cache, ohne Metadaten

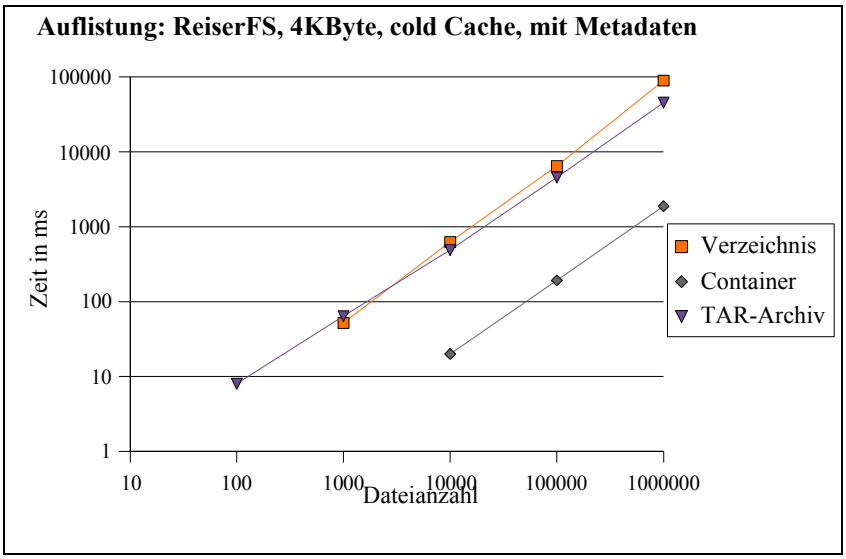


Diagramm 5.9: Auflistung des Inhalts ReiserFS, 4KByte, cold Cache, mit Metadaten

Für Verzeichnisse mit bis zu 1.000 4 KByte-Dateien konnten nur Verringerungen der benötigten Zeiten im niedrigen einstelligen Prozentbereich ermittelt werden. Ab 10.000 4 KByte-Dateien jedoch wurden Inhalte der Verzeichnisse mit und ohne Metadaten der Dateien bei leerem Cache bis zu zehn mal so schnell, wie auf Ext3 gemessen. Mindestens wurde ein Faktor 2 erreicht.

Die Zeiten, die für 4 KByte-Dateien bei gefülltem Cache ermittelt wurden zeigt das folgende Diagramm. Hier benötigt das Listen des Verzeichnisinhalts um Faktor 2-3 weniger Zeit als auf dem Ext3 Dateisystem.

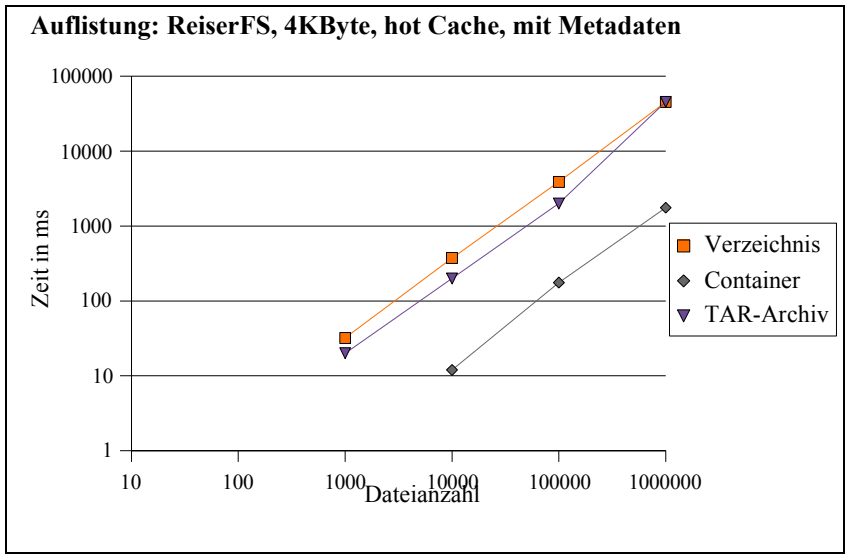


Diagramm 5.10: Auflistung des Inhalts ReiserFS, 4 KByte, hot Cache, mit Metadaten

Damit zeichnet sich folgende Tendenz ab. Das Auflisten von Verzeichnisinhalten benötigt auf einem ReiserFS Dateisystem grundsätzlich deutlich weniger Zeit. Bei kleinen enthaltenen Dateien konnte dieses Verhalten, beim Auflisten mit und ohne Metadaten, durch die Tests belegt werden. Auch für große Dateien gilt diese Tendenz, wenn die Metadaten

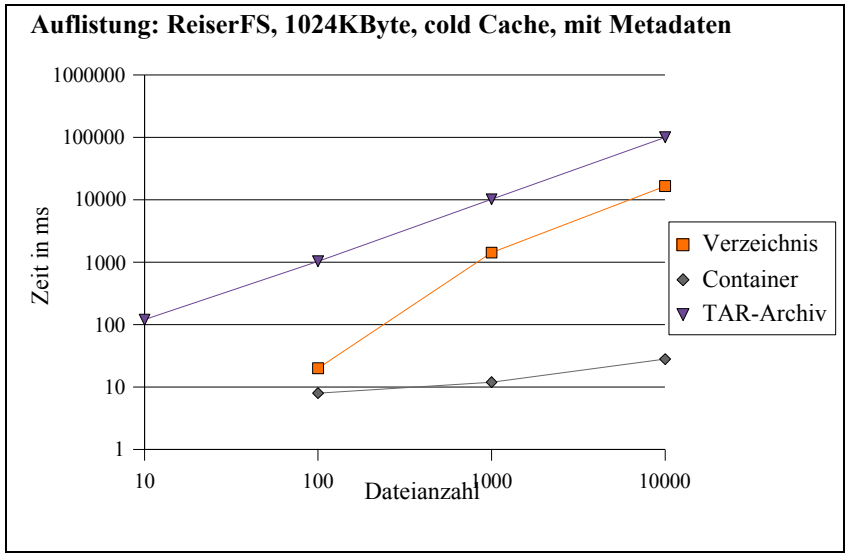


Diagramm 5.11: Auflistung des Inhalts ReiserFS, 1 MByte, cold Cache mit Metadaten

nicht ausgegeben werden müssen. Sollen jedoch mehrere hundert große, in einem Verzeichnis enthaltene Dateien mit deren Metadaten gelistet werden, wird dafür fast die zehnfache Zeit, wie auf Ext3 benötigt. Bei gefülltem Cache hingegen, hat ReiserFS auch bei großen Dateien mit Metadaten keinen Nachteil gegenüber Ext3.

Das Verzeichnis wurde für ReiserFS so ausführlich betrachtet, da weder das Auflisten des TAR-Archivs-Inhalts noch das Auflisten des Containerinhalts von der Verwendung von ReiserFS profitieren. Es konnten auch keine Nachteile festgestellt werden. Die Begründung dafür ist offensichtlich. Da sowohl Containerdatei als auch TAR-Archiv die Metadaten ihrer enthaltenen Datei selbst verwalten, können sie auch nicht von der optimierten Metadatenverwaltung von ReiserFS profitieren.

Zusammenfassend wird festgehalten, dass die Verwendung von Containerdateien bei allen 20 Testfällen sowohl auf Ext3 als auch auch ReiserFS beim Auflisten der enthaltenen Dateien mit und ohne deren Metadaten Leistungsvorteile gegenüber der Verwendung von Verzeichnissen und TAR-Archiven bringt. Für Verzeichnisse reichen diese Vorteile von 50% bei kleinen Dateianzahlen bis zu einem Faktor 500 bei großen Dateianzahlen großer Dateien. Gegenüber TAR-Archiven konnten weit größere Steigerungen gemessen werden.

5.4 Test: Dateizugriff

Ziel des folgenden Tests ist die Evaluation, ob durch die Verwenden einer Containerdatei der Zugriff auf enthaltene Dateien zeitliche Nachteile mit sich bringt. Getestet wird daher wie viel Zeit notwendig ist, um die Nutzdaten einer Datei zu lesen. Dabei wird einerseits auf Dateien in einem Verzeichnis und andererseits auf Dateien in einem Container zugegriffen. Auf den Zugriff auf Dateien, die in einem TAR-Archiv archiviert sind wird verzichtet. Die gemessenen Zeiten werden anschließend einander gegenübergestellt.

Die Dateizugriffe, die in diesem Test durchgeführt werden entsprechen für eine Containerdatei dem zukünftigen Verwendungszweck. Dieser sieht vor, dass aus einer Anwendung heraus auf Containerdateien als Datenquelle zugegriffen werden kann. Dazu wird in diesem Test mit Hilfe der in Kapitel 4 definierten API auf Dateien im Container lesend zugegriffen. Zu diesem Zweck wurde eine Anwendung *apiCTest* implementiert. Diese Anwendung lokalisiert alle Dateien innerhalb einer Containerdatei und liest die gesamten Nutzdaten der Dateien. Zum Vergleich wurde eine zweite Anwendung *apiVTest* implementiert, die für selben Aufgabe die benötigten Zeiten auf Dateien innerhalb eines Verzeichnisses misst.

5.4.1 Testablauf: Dateizugriff

Auch dieser Test baut auf den 20 Testfällen auf, die für die Erstellung von Verzeichnis,

Container und TAR-Archiv definiert wurden. Für jeden dieser Testfälle wird jeweils fünf mal mit leerem Cache und fünf mal mit vollem Cache getestet, wie lange der lesende Zugriff auf Dateien dauert. Dazu lesen die Anwendungen *apiCTest* auf die Nutzdaten der Dateien im Container und *apiVTest* die Nutzdaten der Dateien im Verzeichnis. Um Reproduzierbarkeit der Messergebnisse zu erreichen wird zum leeren des Caches das Dateisystem mit *umount* und *mount* aus und wieder eingehängt.

5.4.2 Auswertung: Dateizugriff

Das erste Diagramm stellt die Testfälle 1-6 dar. Es zeigt damit wie viel Zeit nötig ist, um die Nutzdaten aller 4 KByte Dateien im Verzeichnis und Container auszulesen. Da diese Testfälle mit leerem Cache durchgeführt werden sind jeweils die Metadaten zu den Dateien und die Nutzdaten von der Festplatte zu liefern. Die gemessenen Werte zeigen, dass das Lesen der Dateien aus einem Verzeichnis zehn mal so viel Zeit in Anspruch nimmt wie das Lesen der Dateien aus einem Container. Da die Menge der Nutzdaten in beiden Fällen gleich ist, muss der Zeitunterschied seine Ursache in der Aufwendigkeit der Metadatenoperationen haben, die nötig sind um die Datei zu lokalisieren.

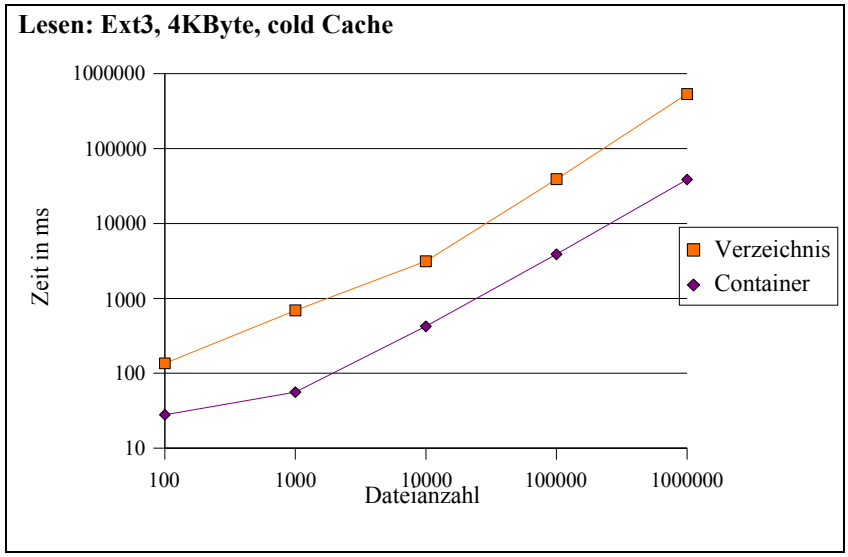


Diagramm 5.12: Lesen der Dateien Ext3, 4 KByte, cold Cache

Im Dateisystem muss dafür mit Hilfe des Dateinamen nach dem passenden *Dentry*-Eintrag gesucht werden. Dieser liegt in einem Datenblock des Verzeichnisses. Mit den daraus extrahierten Informationen kann das entsprechende *Inode* der Datei lokalisiert werden, um anschließend die Nutzdaten aus deren Datenblöcken zu lesen. Beim Container liegen nach einmaliger Lesen des Inhaltsverzeichnisses als Metadaten der enthaltenen Datei vor. Trotzdem auch hier vor dem Dateizugriff die Lokation der Datei im Container festgestellt werden muss benötigt der gesamte Vorgang eine ganze Größenordnung weniger Zeit. Der Vorteil ergibt sich aus der Tatsache, das die dafür verwendete binäre Suche auf Daten zurückgreifen kann, die bereits im Hauptspeicher vorliegen. Beim Verzeichnis sind für das

beschaffen der Metadaten zeitaufwendigere Festplattenzugriffe nötig.

Die im Folgenden dargestellten Messungen bei gefülltem Cache belegen diese Fakten. Da, solange die Metadaten der Dateien und deren Nutzdaten im Cache Platz finden, sich die Zeiten für Dateien im Verzeichnis um den Faktor 10 verringern. Da die Containerdatei auch vom Cache profitiert, können die Dateien trotzdem noch 50% schneller gelesen

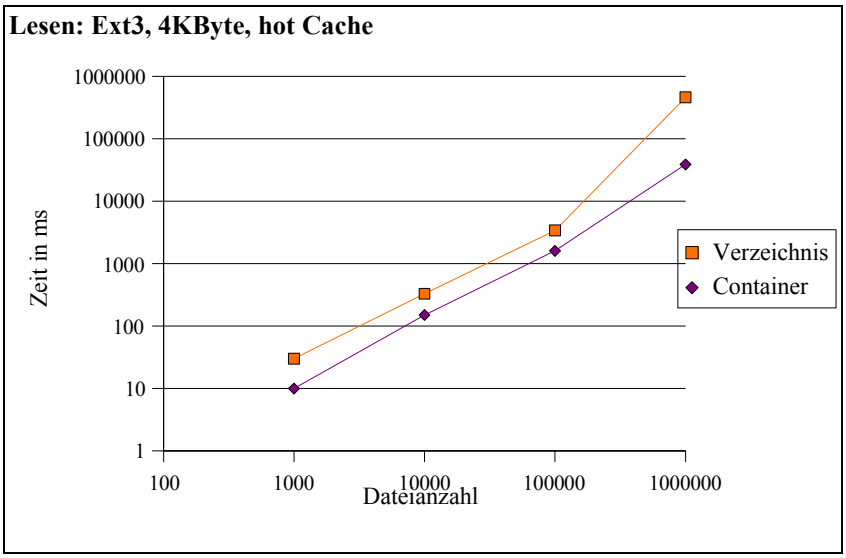


Diagramm 5.13: Lesen der Dateien Ext3, 4 KByte, hot Cache

werden als das beim Verzeichnis der Fall ist. Bei einer Million Dateien passen nicht alle Meta- und Nutzdaten mehr in den Cache. Daher werden hier Zeiten gemessen, die sich nur unwesentlich von denen mit leerem Cache unterscheiden.

Für große Dateien gelten die selben Beobachtungen, wie die folgenden Diagramme belegen.

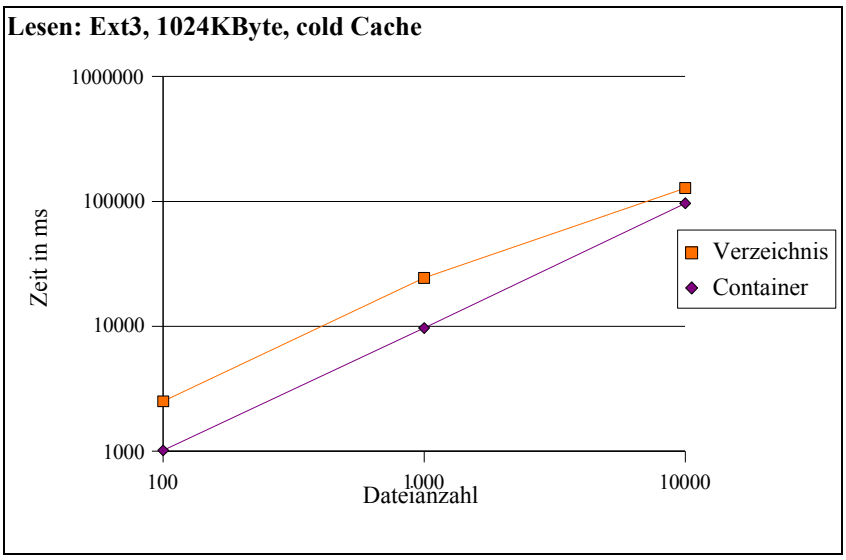


Diagramm 5.14: Lesen der Dateien Ext3, 1024 KByte, cold Cache

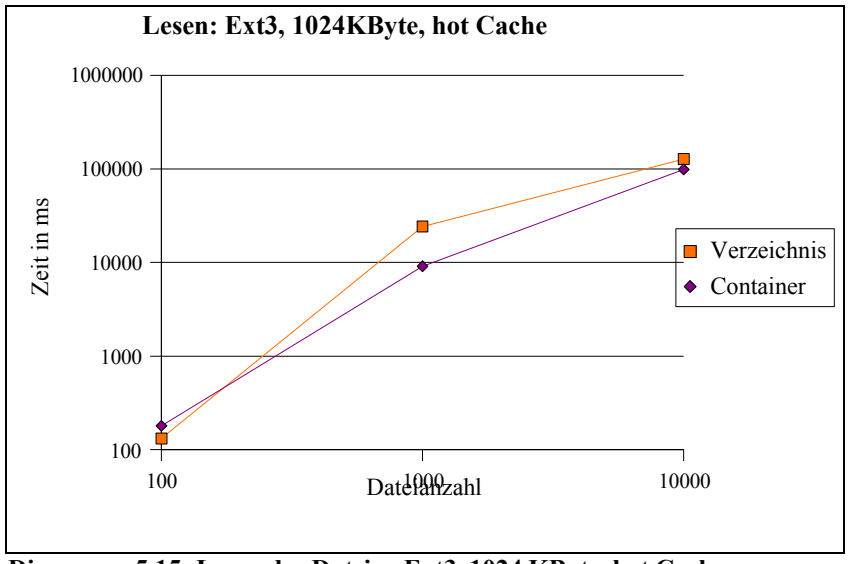


Diagramm 5.15: Lesen der Dateien Ext3, 1024 KByte, hot Cache

Die Zeiten für Container und Verzeichnis liegen zwar grundsätzlich näher beieinander, dass hat aber seine Ursache in der Tatsache, dass das eigentliche Lesen der Dateien ein wesentlich größeren Bestandteil der gemessenen Zeit ausmacht. Der zweite Unterschied ist, dass schon bei 1.000 Dateien nicht mehr alle Nutz- und Metadaten der Dateien in den Cache passen und damit die gemessenen Zeiten auf des Niveau der Testläufe ohne Cache absinken. Die beiden folgenden Diagramme zeigen die dafür gemessenen Zeiten bei leerem und gefülltem Cache.

Auf dem Dateisystem ReiserFS zeichnet sich genau wie in den vorangegangen Test ab, dass, aufgrund der effizienteren Metadatenverwaltung, vor allem das Lesen der Dateien aus Verzeichnissen weniger Zeit als auf Ext3 benötigt. Im Gegensatz dazu kann das Lesen der Dateien im Container nicht davon profitieren, da hier die Metadatenverwaltung nicht vom Dateisystem abhängt. Die folgenden Diagramme illustrieren die gemessenen Zeiten auf ReiserFS.

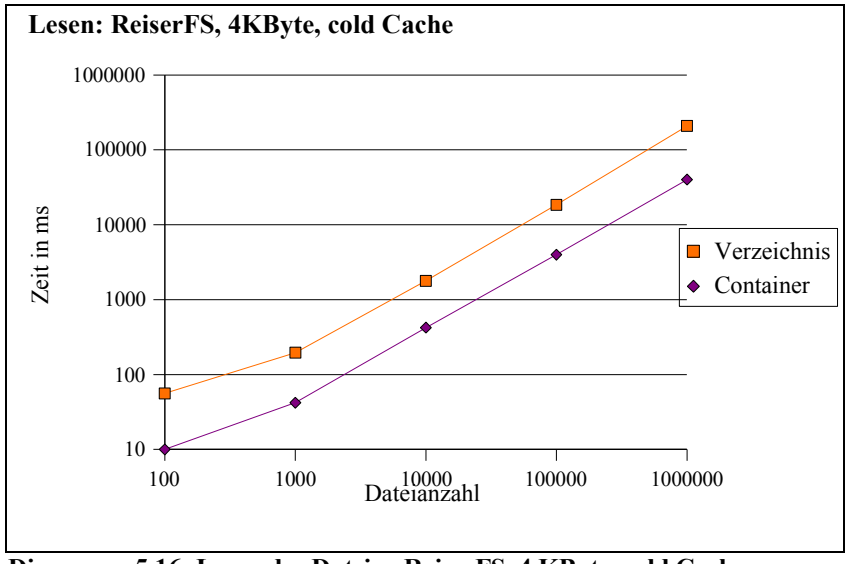


Diagramm 5.16: Lesen der Dateien ReiserFS, 4 KByte, cold Cache

Bei leerem Cache ist damit das Lesen der Dateien aus einem Container um den Faktor 5 schneller. Während bei gefülltem Cache das Lesen aus dem Verzeichnis doppelt so viel Zeit benötigt, wie das Lesen aus einer Containerdatei. Die Ausnahme bilden bei gefülltem Cache die Fälle bei denen nicht alle Meta- und Nutzdaten der zu lesenden Dateien in den Cache passen. Deren Zeiten befinden sich wie schon bei Ext3 auf cold-Cache-Niveau.

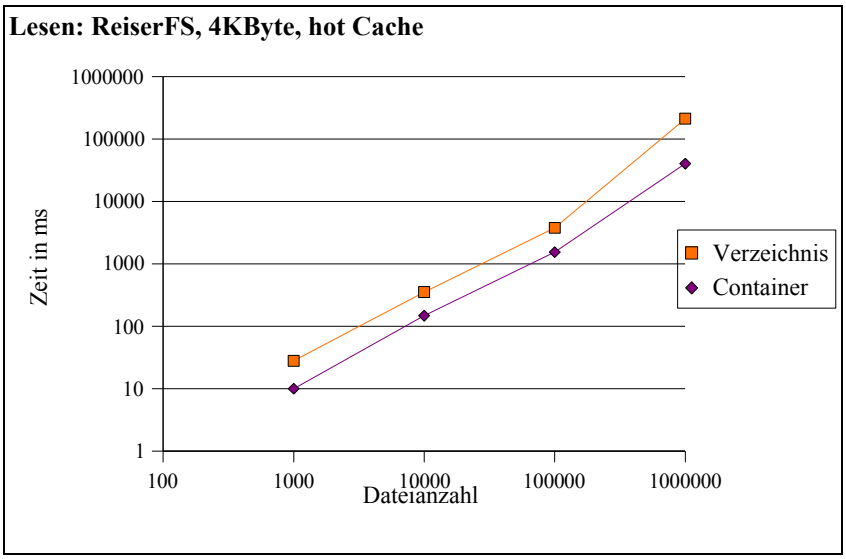


Diagramm 5.17: Lesen der Dateien ReiserFS, 4 KByte, hot Cache

Bei großen Datei sind die gemessenen Zeiten für Verzeichnisse und Container nahezu identisch, da die Zeit für das eigentliche Lesen der Nutzdaten einen noch größeren Teil der Gesamtzeit ausmacht. Die folgenden Abbildungen zeigen die dazu gemessenen Werte.

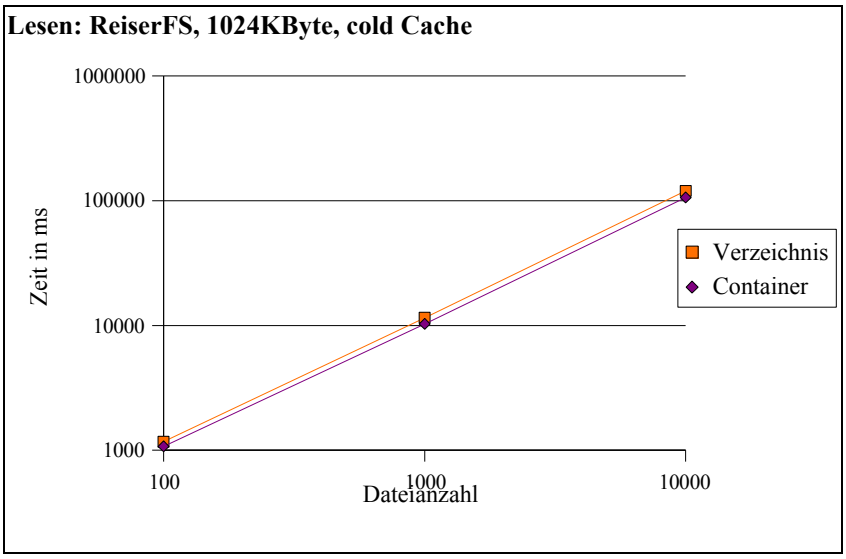


Diagramm 5.18: Lesen der Dateien ReiserFS, 1024 KByte, cold Cache

Auch hier fallen, bei Test mit hot Cache, die Zeiten auf cold-Cache-Niveau sobald nicht mehr alle Nutzdaten und Metadaten der zu lesenden Dateien in den Cachepassen.

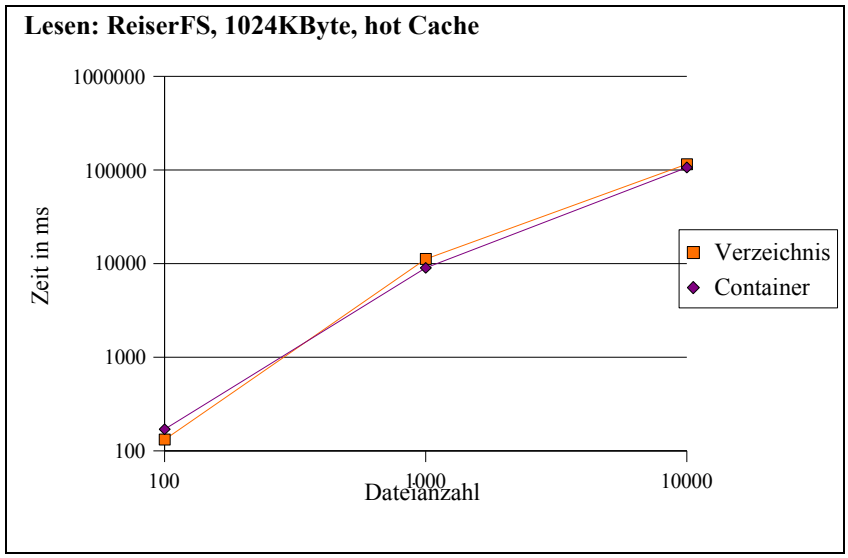


Diagramm 5.19 Lesen der Dateien ReiserFS, 1024 KByte, hot Cache

Damit sind die Testfälle zum Lesen enthaltener Dateien abgedeckt und die Ergebnisse erlauben folgendes Fazit. Das Verwenden der in Kapitel 4 definierten API zum Lesen von Dateien aus einer Containerdatei bringt Leistungsvorteile gegenüber der Verwendung der POSIX-API zum Lesen aus Verzeichnissen. Diese Vorteile entstehen im wesentlich daraus, dass dem Dateisystem die Metadatenverwaltung für die enthaltenen Dateien vorenthalten wird. Daher kann die API für den Container eine eigene effizientere Verwaltung, wie die Testergebnisse belegen, implementieren.

Zusammenfassung / Bewertung

Für den Anwendungsfall bei unserem Forschungspartner Deutsches Krebsforschungszentrum DKFZ wurde in dieser Arbeit ein Containerformat definiert. Der Anwendungsfall sieht vor, eine erhebliche Anzahl von Bilddateien für die Weiterverarbeitung zur Verfügung zu stellen. Die Verwendung dieses Containerformats ermöglicht dabei eine erhebliche Steigerung der Zugriffsleistung gegenüber gängigen Dateisystem-Strukturen.

Zu den hier betrachten gängigen Dateisystemen gehören Ext3 und ReiserFS. Beide entwickeln bei großen Anzahlen von Dateien Leistungsengpässe. Die Gründe dafür liegen in der Tatsache, dass Dateisysteme alle Arten von Dateien unterstützen müssen. Die daraus resultierenden Kompromisse bei der Metadatenverwaltung sind Ursache für die Leistungsmängel.

Die Metadatenverwaltung eines Dateisystems muss für eine Datei grundsätzlich zwei Datenstrukturen verwalten. Einerseits muss ein *Dentry*-Objekt für die Datei vorhanden sein, denn dieses Objekt repräsentiert einen Verzeichniseintrag des Verzeichnisses in dem die Datei liegt und nur in diesem Objekt ist der Dateiname abgelegt. Andererseits muss ein *Inode* für die Datei verwaltet werden in dem die Metadaten, wie Zugriffsrechte und Dateigröße gehalten werden. Der Zugang zu einem *Inode* erfolgt über den assoziierten *Dentry*-Eintrag. Die Nutzdaten einer Datei werden über das *Inode* erreicht oder wie im Falle von ReiserFS direkt im *Inode* abgelegt. Bei großen Dateianzahlen ist das folgende Problem in der Metadatenverwaltung die Hauptursache für die Einbrüche in der Zugriffsleistung. Es kann nicht mehr garantiert werden, dass für jeden Anwendungsfall der Dateizugriff effizient ist, da die räumliche Nähe der Metadaten abnimmt. Das ist der Fall, da für den effizienten Dateizugriff Metadaten und Nutzdaten eng beieinander liegen müssen. Für Verzeichnisse, die auch nur Dateien sind, bedeutet dies, dass alle *Dentrys* möglichst dicht auf dem Datenträger liegen sollten. Für Dateien hingegen sollten *Dentrys*, *Inodes* und Nutzdaten eng zusammen abgelegt sein. Da beides nicht möglich ist, stellen Dateisysteme eigene Optimierungen zu Verfügung, wie die von ReiserFS beschriebene für kleine Dateien. Bei der Entwicklung des Containerformats wurde daher der Reduzierung der Metadatenzugriffe hohe Priorität gewährt.

Bei der Verwendung von Containerdateien ist der Aufwand für die Metadatenverwaltung des darunter liegenden Dateisystems minimal. Die Metadatenzugriffe wurden auf der Ebene des Dateisystems minimiert. Die Metadaten für die im Container enthaltenen Dateien verwaltet nicht das Dateisystem, vielmehr wurde eine API definiert, die die damit im Zusammenhang stehenden Aufgaben erfüllt. Um die Effizienz dieser Verwaltung zu erhöhen wurde das Containerformat so definiert, dass nur zwingend zur Identifizierung und Lokalisation notwendige dateibeschreibende Informationen gespeichert werden. Die Auswahl der entsprechenden Daten folgt direkt aus dem Anwendungsfall am DKFZ. So konnte zum Beispiel auf das Speichern der Dateizugriffsberechtigung verzichtet werden, da die Verzeichnisberechtigungen denen der Dateien entsprechen und daraus folgend auch die Containerzugriffsberechtigungen implizit denen der enthaltenen Dateien entsprechen.

So konnte der Speicherplatz für die notwendigen dateibeschreibenden Daten im Falle von Large File Support auf nur *16 Byte + x Byte* pro Datei reduziert werden. Das *x* entspricht dabei der Länge des Dateinamen. Zudem wurde das ablegen der verbleibenden Metadaten für enthaltene Datei so strukturiert, dass mit hocheffizienten Zugriffsmethoden darauf zugegriffen werden kann. Als Beispiel dafür sei hier das längen-lexikographisch sortierte Inhaltsverzeichnis aufgeführt. Durch diese Sortierung konnte die Binär-Suche als Algorithmus für die Lokalisation der Dateimetadaten verwendet werden. Damit steht ein sehr effizientes Verfahren für den Zugriff auf Dateien im Container zur Verfügung.

Die nachfolgenden Test belegten die Leistungsvorteile, die die Verwendung von Containerdateien mit sich bringt. Es wurde getestet inwiefern sich die Zeiten für die Erstellung eines Verzeichnisses, einer Containerdatei und eines TAR-Archivs unterscheiden. Die Zeiten wurden mit Dateianzahlen zwischen 10 und 1.000.000 und auf den Dateisystemen Ext3 und ReiserFS gemessen. Bei kleinen Dateien konnte so die Leistung des Containers um mindestens Faktor 3 besser als das Verzeichnis dokumentiert werden. Bei großen Dateien konnte das Verzeichnis die Aufgabe schneller abschließen, was aber auf den erhöhten Lese- und Schreibaufwand beim Erstellen von Containerdateien und TAR-Archiven zurückzuführen ist.

Bei den Tests, die der Zeitmessung der Inhaltsauflistung dienten, konnte die Containerdatei in einigen Testfällen die Bearbeitung des Test mehrere hundert Mal schneller als Verzeichnis und TAR-Archiv abschließen. So benötigten Verzeichnis und TAR-Archiv für das Auflisten von einer Millionen Dateien mit deren Metadaten auf dem Dateisystem Ext3 zum Beispiel rund sieben und rund eine Minute, während das Auflisten des Containerinhalts nur rund 2 Sekunden dauerte. Zusätzlich kann die Containerdatei ihre Leistung dateisystemunabhängig entfalten, da die Metadatenverwaltung für die enthaltenen Dateien nicht dem Dateisystem obliegt.

Die Test die der Zeitmessung für das Lesen enthaltener Dateien dienen zeigten, dass auch hier die Verwendung von Containerdateien mit dem in dieser Arbeit definierten Format und der zur Verwendung implementierten API Leistungsvorteile bringt. Auch hier konnte dokumentiert werden, dass das Lesen enthaltener Dateien um eine ganze Größenordnung schneller ist als das Lesen von Dateien innerhalb eines Verzeichnisses. Dies gilt nicht für große Dateien. Bei Dateien ab einer gewissen Größe nimmt der Zeitanteil für Metadatenoperationen im Vergleich zum eigentlichen Lesen der Nutzdaten so stark ab, dass der Zeitvorteil des Containers stark sinkt. Dennoch konnten Dateien aus einem Verzeichnis nicht schneller als aus einem Container gelesen werden.

Durch die Test wurde nachgewiesen, dass das Containerformat in Verbindung mit der implementierten API als Bibliothek *libct.c* und den Kommandozeilen Programmen *ctmk.c*, *ctls.c* und *ctfcp.c* Leistungsvorteile bezüglich der Zugriffseffizienz bietet. Damit bietet die Technologie des in dieser Arbeit definierten Container-Archiv-Formats ein hohes Potenzial für die Weiterentwicklung. Einen Ausblick auf mögliche Erweiterungen und Verwendungsmöglichkeiten liefert das folgende Kapitel.

Ausblick

Die folgende Auflistung gibt einen Ausblick auf zukünftige Möglichkeiten für die Erweiterung der Containerdefinition und zeigt, wie die Verwendungsmöglichkeiten von Containerdateien noch erweitertwerden können.

- Eine Erweiterung der Containerdefinition ist die Erstellung von Containerdateien aus Anwendungen heraus, so dass diese die Ergebnisdateien direkt in der Form einer Containerdatei ablegen. Damit ist eine vorherige Erstellung von Dateien im Verzeichnisbaum überflüssig.
- Zusätzlich ist das Abbilden ganzer Verzeichnisstrukturen in Containerdateien eine mögliche Erweiterung. Damit kann der Container nicht nur eine Menge von Dateien kapseln sondern ganze Teilbäume von Verzeichnisbäumen.
- Um auch Legacy-Anwendungen den Umgang mit Containerdateien zu ermöglichen ohne deren Implementierung verändern zu müssen, ist eine transparente Umsetzung der Schnittstellen für den Zugriff denkbar. Zum Beispiel könnten durch Verwendung einer preloaded library dateibezogenen Befehle gewrappt werden, so dass der Zugriff für höhere Abstraktionsschichten transparent ist.
- Auch das Zulassung von Änderungen der im Container enthaltenen Dateien stellt eine mögliche Erweiterung der Containerdefinition dar. So könnte in einem noch zu definierende Rahmen das Ändern von Dateiinhalten ermöglicht werden, wenn dies nicht der Definition des Container-Formates widerspricht.
- Die Art der Verwendung vom lseek-Systemaufruf in der API-Prozedur ct_file_read() hat zur Folge, dass die oben verwendete Implementierung nicht threadsafe ist. Durch die Verwendung von je einem Datei-Deskriptor pro geöffnete Datei im Container können die daraus resultierenden Probleme mit der Nebenläufigkeit zukünftig ausgeschlossen werden.
- Vorstellbar ist des Weiteren die automatische Erkennung von sequentiellen Dateizugriffen in der Containerdatei. Als Reaktion könnte das System einen sehr viel größeren Teil der Containerdatei von der Festplatte lesen. Im Idealfall wären die benötigten Dateien dann schon vor dem Zugriff im Hauptspeicher verfügbar.

Eine Vielzahl von weiteren Erweiterungen und Verwendungsmöglichkeiten ist denkbar. Abschließend soll eine interessante Möglichkeit erwähnt sein. Sie beschäftigt mit der Möglichkeit der Vollintegration des Container-Formates in ein Dateisystem mit absoluter Transparenz für Benutzerabstraktionsschichten. Interessant ist dabei die Möglichkeit der dezentralen Metadatenverwaltung, da die Container-API die Verwaltung der Metadaten des Inhalts einer Containerdatei übernimmt.

[KL07] Michael Kuhn, Christian Lohse. *Evaluierung paralleler Dateisysteme*, Ruprecht-Karls-Universität Heidelberg, Juni 07.

[FSF07] *GNU tar: an archiver tool*, http://www.gnu.org/software/tar/manual/tar.html, Free Software Foundation, Inc., 2007

[CTT] Rémy Card, Theodore Ts'o, Stephen Tweedie. *Design and Implementation of the Second Extended Filesystem,* http://e2fsprogs.sourceforge.net/ext2intro.html

[BM00] Daniel P. Bovet, Marco Cesati. *Understanding the Linux Kernel*, O'Reilly, Januar 2001

[Mau04] Wolfgang Maurer. *Linux Kernelarchitektur*, Carl Hanser Verlag München Wien, 2004

[Lov05] Robert Love. *Linux-Kernel-Handbuch*, Addison-Wesley Verlag, 2005

[Wik] *ext3*, Wikipedia Die freie Enzyklopädie, http://de.wikipedia.org/wiki/Ext3

[GE05] Richard Gooch, Pekka Engberg. *Overview of the Linux Virtual Filesystem*, /usr/src/linux/Dokumentation/filesystems/vfs.txt, Oktober 2005

[Ext3] *Ext3 Filesystem*, /usr/src/linux/Dokumentation/filesystems/ext3.txt

[Jae05] Andreas Jaeger. *Large File Support in* Linux, http://www.suse.de/~aj/linux_lfs.html, Februar 2005

Anhang A

Testprotokolle

/dev/md2 on /raid/heinrich type ext3 (rw,noexec,nosuid,nodev)
10 (1*4 KB) Dateien on ext3

---

time create_testfiles2.sh V_10:
TIME: 0:00.06 CPU: 35% major: 0 minor: 2875 invol: 19 vol: 28

time ctmk c_10.ct:
TIME: 0:00.00 CPU: 80% major: 0 minor: 134 invol: 0 vol: 23

time tar -cvfV_10:
tar: Removing leading `/' from member names
TIME: 0:00.02 CPU: 0% major: 0 minor: 219 invol: 4 vol: 19

---

time ls -l V_10:

TIME: 0:00.00 CPU: 200% major: 0 minor: 219 invol: 1 vol: 8
TIME: 0:00.00 CPU: 200% major: 0 minor: 219 invol: 4 vol: 3

TIME: 0:00.00 CPU: 0% major: 0 minor: 220 invol: 3 vol: 6
TIME: 0:00.00 CPU: 0% major: 0 minor: 220 invol: 4 vol: 1

TIME: 0:00.00 CPU: 400% major: 0 minor: 218 invol: 5 vol: 4
TIME: 0:00.00 CPU: 200% major: 0 minor: 220 invol: 3 vol: 3

TIME: 0:00.00 CPU: 0% major: 0 minor: 219 invol: 1 vol: 8
TIME: 0:00.00 CPU: 200% major: 0 minor: 220 invol: 5 vol: 1

TIME: 0:00.00 CPU: 0% major: 0 minor: 220 invol: 4 vol: 4
TIME: 0:00.00 CPU: 400% major: 0 minor: 220 invol: 5 vol: 1

10

---

time tar -tf T_10.tar:

TIME: 0:00.00 CPU: 0% major: 0 minor: 197 invol: 1 vol: 5
TIME: 0:00.00 CPU: 0% major: 0 minor: 199 invol: 0 vol: 1

TIME: 0:00.00 CPU: 400% major: 0 minor: 197 invol: 1 vol: 6
TIME: 0:00.00 CPU: 0% major: 0 minor: 197 invol: 0 vol: 1

TIME: 0:00.00 CPU: 200% major: 0 minor: 198 invol: 1 vol: 6
TIME: 0:00.00 CPU: 400% major: 0 minor: 197 invol: 1 vol: 1

TIME: 0:00.00 CPU: 0% major: 0 minor: 198 invol: 0 vol: 6
TIME: 0:00.00 CPU: 0% major: 0 minor: 198 invol: 1 vol: 1

TIME: 0:00.00 CPU: 200% major: 0 minor: 198 invol: 1 vol: 5
TIME: 0:00.00 CPU: 0% major: 0 minor: 197 invol: 0 vol: 1

11

---

time ctls c_10.ct:

TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 1 vol: 9
TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 0 vol: 2

TIME: 0:00.00 CPU: 0% major: 0 minor: 122 invol: 0 vol: 5
TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 0 vol: 2

TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 1 vol: 5
TIME: 0:00.00 CPU: 0% major: 0 minor: 122 invol: 0 vol: 2

TIME: 0:00.00 CPU: 400% major: 0 minor: 121 invol: 1 vol: 5
TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 0 vol: 2

TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 1 vol: 5
TIME: 0:00.00 CPU: 0% major: 0 minor: 122 invol: 0 vol: 2

10

---

time ls -l V_10:

TIME: 0:00.00 CPU: 133% major: 0 minor: 237 invol: 3 vol: 6
TIME: 0:00.00 CPU: 0% major: 0 minor: 238 invol: 2 vol: 3

TIME: 0:00.00 CPU: 0% major: 0 minor: 240 invol: 5 vol: 4
TIME: 0:00.00 CPU: 0% major: 0 minor: 238 invol: 4 vol: 1

TIME: 0:00.00 CPU: 0% major: 0 minor: 239 invol: 5 vol: 4
TIME: 0:00.00 CPU: 200% major: 0 minor: 237 invol: 3 vol: 5

TIME: 0:00.00 CPU: 0% major: 0 minor: 239 invol: 3 vol: 6
TIME: 0:00.00 CPU: 200% major: 0 minor: 237 invol: 1 vol: 5

TIME: 0:00.00 CPU: 0% major: 0 minor: 239 invol: 5 vol: 4
TIME: 0:00.00 CPU: 200% major: 0 minor: 238 invol: 3 vol: 5

11

/dev/md2 on /raid/heinrich type reiserfs (rw,noexec,nosuid,nodev)
10 (1*4 KB) Dateien on reiserfs

---

time create_testfiles2.sh V_10:
TIME: 0:00.36 CPU: 6% major: 7 minor: 2853 invol: 24 vol: 48

time ctmk c_10.ct:
TIME: 0:00.00 CPU: 133% major: 0 minor: 132 invol: 1 vol: 18

time tar -cvfV_10:
tar: Removing leading `/' from member names
TIME: 0:00.08 CPU: 4% major: 4 minor: 215 invol: 3 vol: 22

---

time ls -l V_10:

TIME: 0:00.00 CPU: 0% major: 1 minor: 217 invol: 3 vol: 6
TIME: 0:00.00 CPU: 200% major: 0 minor: 221 invol: 5 vol: 1

TIME: 0:00.00 CPU: 0% major: 0 minor: 220 invol: 3 vol: 5
TIME: 0:00.00 CPU: 200% major: 0 minor: 220 invol: 3 vol: 3

TIME: 0:00.00 CPU: 0% major: 0 minor: 220 invol: 4 vol: 1
TIME: 0:00.00 CPU: 200% major: 0 minor: 219 invol: 5 vol: 1

TIME: 0:00.00 CPU: 0% major: 0 minor: 220 invol: 4 vol: 3
TIME: 0:00.00 CPU: 200% major: 0 minor: 220 invol: 5 vol: 1

TIME: 0:00.00 CPU: 0% major: 0 minor: 220 invol: 3 vol: 3
TIME: 0:00.00 CPU: 200% major: 0 minor: 219 invol: 3 vol: 4

10

---

time tar -tf T_10.tar:

TIME: 0:00.00 CPU: 0% major: 0 minor: 197 invol: 1 vol: 3
TIME: 0:00.00 CPU: 400% major: 0 minor: 197 invol: 1 vol: 1

TIME: 0:00.00 CPU: 0% major: 0 minor: 196 invol: 1 vol: 3
TIME: 0:00.00 CPU: 400% major: 0 minor: 197 invol: 1 vol: 1

TIME: 0:00.00 CPU: 0% major: 0 minor: 198 invol: 1 vol: 3
TIME: 0:00.00 CPU: 400% major: 0 minor: 198 invol: 1 vol: 1

TIME: 0:00.00 CPU: 0% major: 0 minor: 197 invol: 1 vol: 4
TIME: 0:00.00 CPU: 400% major: 0 minor: 197 invol: 1 vol: 1

TIME: 0:00.00 CPU: 200% major: 0 minor: 197 invol: 0 vol: 3
TIME: 0:00.00 CPU: 0% major: 0 minor: 197 invol: 0 vol: 1

11

---

time etls c_10.ct:

TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 0 vol: 5
TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 1 vol: 2

TIME: 0:00.00 CPU: 0% major: 0 minor: 122 invol: 0 vol: 4
TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 1 vol: 2

TIME: 0:00.00 CPU: 0% major: 0 minor: 120 invol: 1 vol: 4
TIME: 0:00.00 CPU: 0% major: 0 minor: 122 invol: 1 vol: 2

TIME: 0:00.00 CPU: 400% major: 0 minor: 122 invol: 0 vol: 4
TIME: 0:00.00 CPU: 0% major: 0 minor: 122 invol: 1 vol: 2

TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 0 vol: 4
TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 1 vol: 2

10

---

time ls -l V_10:

TIME: 0:00.00 CPU: 133% major: 0 minor: 239 invol: 4 vol: 3
TIME: 0:00.00 CPU: 200% major: 0 minor: 239 invol: 4 vol: 1

TIME: 0:00.00 CPU: 133% major: 0 minor: 239 invol: 3 vol: 5
TIME: 0:00.00 CPU: 200% major: 0 minor: 238 invol: 5 vol: 1

TIME: 0:00.00 CPU: 133% major: 0 minor: 240 invol: 5 vol: 1
TIME: 0:00.00 CPU: 0% major: 0 minor: 238 invol: 3 vol: 5

TIME: 0:00.00 CPU: 133% major: 0 minor: 238 invol: 6 vol: 1
TIME: 0:00.00 CPU: 200% major: 0 minor: 239 invol: 4 vol: 3

TIME: 0:00.00 CPU: 0% major: 0 minor: 238 invol: 3 vol: 3
TIME: 0:00.00 CPU: 200% major: 0 minor: 238 invol: 1 vol: 5

11

```
---------------------------------------------------------------------------------------------

time tar -tvf T_10.tar:

TIME: 0:00.00   CPU: 0%         major: 0        minor: 209      invol: 1        vol: 5
TIME: 0:00.00   CPU: 200%       major: 0        minor: 209      invol: 1        vol: 1

TIME: 0:00.00   CPU: 0%         major: 0        minor: 209      invol: 0        vol: 5
TIME: 0:00.00   CPU: 0%         major: 0        minor: 208      invol: 0        vol: 1

TIME: 0:00.00   CPU: 200%       major: 0        minor: 208      invol: 1        vol: 4
TIME: 0:00.00   CPU: 0%         major: 0        minor: 210      invol: 0        vol: 1

TIME: 0:00.00   CPU: 0%         major: 0        minor: 207      invol: 1        vol: 5
TIME: 0:00.00   CPU: 0%         major: 0        minor: 209      invol: 0        vol: 1

TIME: 0:00.00   CPU: 0%         major: 0        minor: 208      invol: 1        vol: 5
TIME: 0:00.00   CPU: 0%         major: 0        minor: 208      invol: 0        vol: 1

11
---------------------------------------------------------------------------------------------

time ctls -l c_10.ct:

TIME: 0:00.00   CPU: 0%         major: 0        minor: 148      invol: 2        vol: 7
TIME: 0:00.00   CPU: 0%         major: 0        minor: 150      invol: 5        vol: 2

TIME: 0:00.00   CPU: 400%       major: 0        minor: 150      invol: 3        vol: 4
TIME: 0:00.00   CPU: 400%       major: 0        minor: 149      invol: 5        vol: 2

TIME: 0:00.00   CPU: 0%         major: 0        minor: 149      invol: 3        vol: 4
TIME: 0:00.00   CPU: 400%       major: 0        minor: 148      invol: 3        vol: 4

TIME: 0:00.00   CPU: 400%       major: 0        minor: 150      invol: 5        vol: 2
TIME: 0:00.00   CPU: 400%       major: 0        minor: 150      invol: 4        vol: 4

TIME: 0:00.00   CPU: 400%       major: 0        minor: 150      invol: 3        vol: 4
TIME: 0:00.00   CPU: 400%       major: 0        minor: 149      invol: 5        vol: 2

10
---------------------------------------------------------------------------------------------

/dev/md2 on /raid/heinrich type ext3 (rw,noexec,nosuid,nodev)
100 (1*4 KB) Dateien on ext3
---------------------------------------------------------------------------------------------

time create_testfiles2.sh V_100:
TIME: 0:00.22   CPU: 82%        major: 0        minor: 22115    invol: 177      vol: 185

time ctmk c_100.ct:
TIME: 0:00.02   CPU: 34%        major: 0        minor: 134      invol: 1        vol: 110

time tar -cvf V_100:
tar: Removing leading `/' from member names
TIME: 0:00.21   CPU: 7%         major: 0        minor: 219      invol: 4        vol: 112
---------------------------------------------------------------------------------------------

time ls -l V_100:

TIME: 0:00.05   CPU: 7%         major: 0        minor: 225      invol: 1        vol: 7
TIME: 0:00.00   CPU: 0%         major: 0        minor: 225      invol: 2        vol: 4

TIME: 0:00.00   CPU: 0%         major: 0        minor: 224      invol: 5        vol: 4
TIME: 0:00.00   CPU: 0%         major: 0        minor: 225      invol: 3        vol: 2

TIME: 0:00.00   CPU: 133%       major: 0        minor: 225      invol: 0        vol: 8
TIME: 0:00.00   CPU: 0%         major: 0        minor: 225      invol: 2        vol: 4

TIME: 0:00.00   CPU: 0%         major: 0        minor: 225      invol: 5        vol: 4
TIME: 0:00.00   CPU: 0%         major: 0        minor: 225      invol: 1        vol: 5

TIME: 0:00.00   CPU: 0%         major: 0        minor: 225      invol: 2        vol: 8
TIME: 0:00.00   CPU: 200%       major: 0        minor: 224      invol: 5        vol: 1

100
---------------------------------------------------------------------------------------------

time tar -tf T_100.tar:

TIME: 0:00.00   CPU: 57%        major: 0        minor: 197      invol: 1        vol: 8
TIME: 0:00.00   CPU: 0%         major: 0        minor: 197      invol: 0        vol: 1

TIME: 0:00.03   CPU: 0%         major: 0        minor: 198      invol: 1        vol: 7
TIME: 0:00.00   CPU: 133%       major: 0        minor: 198      invol: 1        vol: 1

TIME: 0:00.02   CPU: 0%         major: 0        minor: 198      invol: 1        vol: 7
TIME: 0:00.00   CPU: 133%       major: 0        minor: 197      invol: 1        vol: 1

TIME: 0:00.03   CPU: 0%         major: 0        minor: 199      invol: 0        vol: 7
TIME: 0:00.00   CPU: 133%       major: 0        minor: 197      invol: 1        vol: 1

TIME: 0:00.02   CPU: 14%        major: 0        minor: 197      invol: 1        vol: 7
TIME: 0:00.00   CPU: 133%       major: 0        minor: 198      invol: 2        vol: 1

101
---------------------------------------------------------------------------------------------

time ctls c_100.ct:

TIME: 0:00.03   CPU: 12%        major: 0        minor: 122      invol: 0        vol: 6
TIME: 0:00.00   CPU: 0%         major: 0        minor: 122      invol: 0        vol: 2

TIME: 0:00.00   CPU: 200%       major: 0        minor: 121      invol: 0        vol: 6
TIME: 0:00.00   CPU: 0%         major: 0        minor: 121      invol: 0        vol: 2

TIME: 0:00.00   CPU: 0%         major: 0        minor: 121      invol: 0        vol: 6
TIME: 0:00.00   CPU: 0%         major: 0        minor: 122      invol: 0        vol: 2

TIME: 0:00.00   CPU: 0%         major: 0        minor: 122      invol: 1        vol: 6
```

```
---------------------------------------------------------------------------------------------

time tar -tvf T_10.tar:

TIME: 0:00.00   CPU: 200%       major: 0        minor: 208      invol: 1        vol: 3
TIME: 0:00.00   CPU: 0%         major: 0        minor: 208      invol: 0        vol: 1

TIME: 0:00.00   CPU: 0%         major: 0        minor: 208      invol: 1        vol: 3
TIME: 0:00.00   CPU: 200%       major: 0        minor: 208      invol: 1        vol: 1

TIME: 0:00.00   CPU: 0%         major: 0        minor: 209      invol: 1        vol: 3
TIME: 0:00.00   CPU: 200%       major: 0        minor: 208      invol: 1        vol: 1

TIME: 0:00.00   CPU: 0%         major: 0        minor: 209      invol: 1        vol: 3
TIME: 0:00.00   CPU: 0%         major: 0        minor: 209      invol: 1        vol: 1

TIME: 0:00.00   CPU: 200%       major: 0        minor: 208      invol: 1        vol: 3
TIME: 0:00.00   CPU: 200%       major: 0        minor: 209      invol: 1        vol: 1

11
---------------------------------------------------------------------------------------------

time ctls -l c_10.ct:

TIME: 0:00.00   CPU: 200%       major: 0        minor: 150      invol: 4        vol: 6
TIME: 0:00.00   CPU: 0%         major: 0        minor: 149      invol: 1        vol: 6

TIME: 0:00.00   CPU: 0%         major: 0        minor: 149      invol: 3        vol: 4
TIME: 0:00.00   CPU: 0%         major: 0        minor: 149      invol: 2        vol: 4

TIME: 0:00.00   CPU: 0%         major: 0        minor: 149      invol: 1        vol: 6
TIME: 0:00.00   CPU: 0%         major: 0        minor: 148      invol: 2        vol: 4

TIME: 0:00.00   CPU: 0%         major: 0        minor: 150      invol: 4        vol: 4
TIME: 0:00.00   CPU: 0%         major: 0        minor: 150      invol: 4        vol: 2

TIME: 0:00.00   CPU: 0%         major: 0        minor: 150      invol: 3        vol: 4
TIME: 0:00.00   CPU: 0%         major: 0        minor: 150      invol: 2        vol: 4

10
---------------------------------------------------------------------------------------------

/dev/md2 on /raid/heinrich type reiserfs (rw,noexec,nosuid,nodev)
100 (1*4 KB) Dateien on reiserfs
---------------------------------------------------------------------------------------------

time create_testfiles2.sh V_100:
TIME: 0:00.22   CPU: 84%        major: 0        minor: 22304    invol: 185      vol: 190

time ctmk c_100.ct:
TIME: 0:00.02   CPU: 69%        major: 0        minor: 134      invol: 1        vol: 106

time tar -cvf V_100:
tar: Removing leading `/' from member names
TIME: 0:00.05   CPU: 27%        major: 0        minor: 219      invol: 5        vol: 107
---------------------------------------------------------------------------------------------

time ls -l V_100:

TIME: 0:00.00   CPU: 200%       major: 0        minor: 223      invol: 5        vol: 1
TIME: 0:00.00   CPU: 200%       major: 0        minor: 222      invol: 3        vol: 4

TIME: 0:00.00   CPU: 200%       major: 0        minor: 224      invol: 3        vol: 2
TIME: 0:00.00   CPU: 200%       major: 0        minor: 223      invol: 5        vol: 1

TIME: 0:00.00   CPU: 200%       major: 0        minor: 224      invol: 3        vol: 2
TIME: 0:00.00   CPU: 200%       major: 0        minor: 223      invol: 1        vol: 5

TIME: 0:00.00   CPU: 200%       major: 0        minor: 223      invol: 4        vol: 1
TIME: 0:00.00   CPU: 200%       major: 0        minor: 224      invol: 5        vol: 1

TIME: 0:00.00   CPU: 200%       major: 0        minor: 224      invol: 4        vol: 1
TIME: 0:00.00   CPU: 200%       major: 0        minor: 224      invol: 4        vol: 3

100
---------------------------------------------------------------------------------------------

time tar -tf T_100.tar:

TIME: 0:00.00   CPU: 57%        major: 0        minor: 198      invol: 1        vol: 4
TIME: 0:00.00   CPU: 133%       major: 0        minor: 198      invol: 1        vol: 1

TIME: 0:00.00   CPU: 0%         major: 0        minor: 198      invol: 1        vol: 4
TIME: 0:00.00   CPU: 133%       major: 0        minor: 198      invol: 1        vol: 1

TIME: 0:00.00   CPU: 0%         major: 0        minor: 198      invol: 1        vol: 4
TIME: 0:00.00   CPU: 133%       major: 0        minor: 198      invol: 1        vol: 1

TIME: 0:00.00   CPU: 0%         major: 0        minor: 197      invol: 1        vol: 4
TIME: 0:00.00   CPU: 133%       major: 0        minor: 197      invol: 1        vol: 1

TIME: 0:00.00   CPU: 171%       major: 0        minor: 198      invol: 1        vol: 4
TIME: 0:00.00   CPU: 133%       major: 0        minor: 198      invol: 1        vol: 1

101
---------------------------------------------------------------------------------------------

time ctls c_100.ct:

TIME: 0:00.00   CPU: 0%         major: 0        minor: 122      invol: 1        vol: 5
TIME: 0:00.00   CPU: 0%         major: 0        minor: 120      invol: 1        vol: 2

TIME: 0:00.00   CPU: 0%         major: 0        minor: 122      invol: 0        vol: 5
TIME: 0:00.00   CPU: 0%         major: 0        minor: 121      invol: 1        vol: 2

TIME: 0:00.00   CPU: 400%       major: 0        minor: 121      invol: 0        vol: 5
TIME: 0:00.00   CPU: 0%         major: 0        minor: 122      invol: 1        vol: 2

TIME: 0:00.00   CPU: 200%       major: 0        minor: 121      invol: 1        vol: 5
```

TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 0 vol: 2	TIME: 0:00.00 CPU: 0% major: 0 minor: 120 invol: 1 vol: 2
TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 0 vol: 6	TIME: 0:00.00 CPU: 0% major: 0 minor: 122 invol: 0 vol: 5
TIME: 0:00.00 CPU: 0% major: 0 minor: 120 invol: 1 vol: 2	TIME: 0:00.00 CPU: 0% major: 0 minor: 122 invol: 1 vol: 2
100	100
---	---
time ls -l V_100:	time ls -l V_100:
TIME: 0:00.04 CPU: 33% major: 0 minor: 243 invol: 5 vol: 7	TIME: 0:00.00 CPU: 114% major: 0 minor: 242 invol: 1 vol: 8
TIME: 0:00.00 CPU: 133% major: 0 minor: 243 invol: 3 vol: 3	TIME: 0:00.00 CPU: 133% major: 0 minor: 242 invol: 3 vol: 3
TIME: 0:00.00 CPU: 50% major: 0 minor: 243 invol: 4 vol: 7	TIME: 0:00.00 CPU: 114% major: 0 minor: 243 invol: 5 vol: 4
TIME: 0:00.00 CPU: 66% major: 0 minor: 244 invol: 3 vol: 3	TIME: 0:00.00 CPU: 66% major: 0 minor: 242 invol: 4 vol: 3
TIME: 0:00.00 CPU: 100% major: 0 minor: 244 invol: 3 vol: 9	TIME: 0:00.00 CPU: 100% major: 0 minor: 242 invol: 5 vol: 4
TIME: 0:00.00 CPU: 133% major: 0 minor: 243 invol: 5 vol: 1	TIME: 0:00.00 CPU: 133% major: 0 minor: 243 invol: 1 vol: 5
TIME: 0:00.00 CPU: 100% major: 0 minor: 243 invol: 2 vol: 11	TIME: 0:00.00 CPU: 114% major: 0 minor: 242 invol: 5 vol: 4
TIME: 0:00.00 CPU: 133% major: 0 minor: 243 invol: 3 vol: 3	TIME: 0:00.00 CPU: 133% major: 0 minor: 244 invol: 5 vol: 1
TIME: 0:00.00 CPU: 100% major: 0 minor: 243 invol: 5 vol: 7	TIME: 0:00.00 CPU: 114% major: 0 minor: 242 invol: 5 vol: 4
TIME: 0:00.00 CPU: 133% major: 0 minor: 242 invol: 2 vol: 5	TIME: 0:00.00 CPU: 133% major: 0 minor: 243 invol: 1 vol: 5
101	101
---	---
time tar -tvf T_100.tar:	time tar -tvf T_100.tar:
TIME: 0:00.04 CPU: 0% major: 0 minor: 208 invol: 0 vol: 7	TIME: 0:00.00 CPU: 50% major: 0 minor: 209 invol: 1 vol: 4
TIME: 0:00.00 CPU: 133% major: 0 minor: 208 invol: 0 vol: 1	TIME: 0:00.00 CPU: 133% major: 0 minor: 209 invol: 0 vol: 1
TIME: 0:00.04 CPU: 0% major: 0 minor: 207 invol: 0 vol: 7	TIME: 0:00.01 CPU: 0% major: 0 minor: 208 invol: 1 vol: 5
TIME: 0:00.00 CPU: 266% major: 0 minor: 209 invol: 1 vol: 1	TIME: 0:00.00 CPU: 133% major: 0 minor: 208 invol: 1 vol: 1
TIME: 0:00.02 CPU: 16% major: 0 minor: 209 invol: 1 vol: 7	TIME: 0:00.01 CPU: 25% major: 0 minor: 208 invol: 1 vol: 4
TIME: 0:00.00 CPU: 266% major: 0 minor: 209 invol: 1 vol: 1	TIME: 0:00.00 CPU: 133% major: 0 minor: 209 invol: 1 vol: 1
TIME: 0:00.02 CPU: 33% major: 0 minor: 209 invol: 2 vol: 7	TIME: 0:00.01 CPU: 22% major: 0 minor: 209 invol: 1 vol: 4
TIME: 0:00.00 CPU: 133% major: 0 minor: 209 invol: 0 vol: 1	TIME: 0:00.00 CPU: 133% major: 0 minor: 208 invol: 0 vol: 1
TIME: 0:00.02 CPU: 17% major: 0 minor: 208 invol: 2 vol: 7	TIME: 0:00.01 CPU: 23% major: 0 minor: 208 invol: 0 vol: 4
TIME: 0:00.00 CPU: 133% major: 0 minor: 210 invol: 0 vol: 1	TIME: 0:00.00 CPU: 133% major: 0 minor: 209 invol: 1 vol: 1
101	101
---	---
time ctls -l c_100.ct:	time ctls -l c_100.ct:
TIME: 0:00.00 CPU: 0% major: 0 minor: 150 invol: 5 vol: 6	TIME: 0:00.00 CPU: 133% major: 0 minor: 150 invol: 5 vol: 7
TIME: 0:00.00 CPU: 0% major: 0 minor: 152 invol: 4 vol: 2	TIME: 0:00.00 CPU: 200% major: 0 minor: 151 invol: 4 vol: 4
TIME: 0:00.00 CPU: 400% major: 0 minor: 150 invol: 3 vol: 6	TIME: 0:00.00 CPU: 0% major: 0 minor: 150 invol: 3 vol: 6
TIME: 0:00.00 CPU: 0% major: 0 minor: 150 invol: 3 vol: 4	TIME: 0:00.00 CPU: 400% major: 0 minor: 150 invol: 1 vol: 6
TIME: 0:00.00 CPU: 0% major: 0 minor: 150 invol: 2 vol: 4	TIME: 0:00.00 CPU: 0% major: 0 minor: 150 invol: 1 vol: 6
TIME: 0:00.00 CPU: 0% major: 0 minor: 150 invol: 3 vol: 4	TIME: 0:00.00 CPU: 0% major: 0 minor: 150 invol: 2 vol: 6
TIME: 0:00.00 CPU: 0% major: 0 minor: 151 invol: 3 vol: 4	TIME: 0:00.00 CPU: 0% major: 0 minor: 152 invol: 4 vol: 4
TIME: 0:00.00 CPU: 0% major: 0 minor: 149 invol: 4 vol: 4	TIME: 0:00.00 CPU: 0% major: 0 minor: 150 invol: 3 vol: 6
TIME: 0:00.00 CPU: 0% major: 0 minor: 150 invol: 4 vol: 4	TIME: 0:00.00 CPU: 0% major: 0 minor: 150 invol: 1 vol: 6
TIME: 0:00.00 CPU: 0% major: 0 minor: 151 invol: 3 vol: 4	TIME: 0:00.00 CPU: 0% major: 0 minor: 151 invol: 4 vol: 2
100	100
---	---
/dev/md2 on /raid/heinrich type ext3 (rw,noexec,nosuid,nodev) 1000 (1*4 KB) Dateien on ext3	/dev/md2 on /raid/heinrich type reiserfs (rw,noexec,nosuid,nodev) 1000 (1*4 KB) Dateien on reiserfs
---	---
time create_testfiles2.sh V_1000: TIME: 0:01.94 CPU: 92% major: 0 minor: 214650 invol: 1745 vol: 1757	time create_testfiles2.sh V_1000: TIME: 0:02.01 CPU: 69% major: 0 minor: 214581 invol: 1822 vol: 1829
time ctmk c_1000.ct: TIME: 0:00.22 CPU: 61% major: 0 minor: 184 invol: 4 vol: 1046	time ctmk c_1000.ct: TIME: 0:00.23 CPU: 60% major: 0 minor: 183 invol: 1 vol: 1030
time tar -cvf V_1000: tar: Removing leading `/' from member names TIME: 0:03.57 CPU: 3% major: 0 minor: 223 invol: 3 vol: 1050	time tar -cvf V_1000: tar: Removing leading `/' from member names TIME: 0:00.28 CPU: 40% major: 0 minor: 223 invol: 4 vol: 1035
---	---
time ls -l V_1000:	time ls -l V_1000:
TIME: 0:00.01 CPU: 80% major: 0 minor: 308 invol: 4 vol: 14	TIME: 0:00.02 CPU: 34% major: 0 minor: 292 invol: 1 vol: 8
TIME: 0:00.00 CPU: 100% major: 0 minor: 304 invol: 3 vol: 3	TIME: 0:00.00 CPU: 133% major: 0 minor: 290 invol: 3 vol: 3
TIME: 0:00.01 CPU: 40% major: 0 minor: 305 invol: 3 vol: 14	TIME: 0:00.00 CPU: 66% major: 0 minor: 290 invol: 4 vol: 4
TIME: 0:00.00 CPU: 100% major: 0 minor: 308 invol: 2 vol: 2	TIME: 0:00.00 CPU: 66% major: 0 minor: 292 invol: 2 vol: 5
TIME: 0:00.00 CPU: 88% major: 0 minor: 305 invol: 3 vol: 14	TIME: 0:00.00 CPU: 133% major: 0 minor: 291 invol: 4 vol: 4
TIME: 0:00.00 CPU: 100% major: 0 minor: 306 invol: 3 vol: 3	TIME: 0:00.00 CPU: 133% major: 0 minor: 292 invol: 5 vol: 1
TIME: 0:00.00 CPU: 44% major: 0 minor: 307 invol: 2 vol: 14	TIME: 0:00.00 CPU: 133% major: 0 minor: 291 invol: 0 vol: 6
TIME: 0:00.00 CPU: 100% major: 0 minor: 307 invol: 3 vol: 3	TIME: 0:00.00 CPU: 133% major: 0 minor: 292 invol: 1 vol: 5
TIME: 0:00.00 CPU: 44% major: 0 minor: 305 invol: 3 vol: 14	TIME: 0:00.00 CPU: 66% major: 0 minor: 293 invol: 5 vol: 4
TIME: 0:00.00 CPU: 100% major: 0 minor: 306 invol: 4 vol: 3	TIME: 0:00.00 CPU: 133% major: 0 minor: 291 invol: 3 vol: 5
1000	1000
---	---
time tar -tf T_1000.tar:	time tar -tf T_1000.tar:
TIME: 0:00.04 CPU: 46% major: 0 minor: 197 invol: 3 vol: 20	TIME: 0:00.04 CPU: 71% major: 0 minor: 199 invol: 1 vol: 33
TIME: 0:00.01 CPU: 114% major: 0 minor: 197 invol: 1 vol: 1	TIME: 0:00.01 CPU: 114% major: 0 minor: 198 invol: 1 vol: 1
TIME: 0:00.09 CPU: 8% major: 0 minor: 197 invol: 2 vol: 24	TIME: 0:00.05 CPU: 39% major: 0 minor: 197 invol: 1 vol: 23

```
TIME: 0:00.01   CPU: 85%        major: 0        minor: 198      invol: 0        vol: 1

TIME: 0:00.05   CPU: 45%        major: 0        minor: 198      invol: 5        vol: 26
TIME: 0:00.01   CPU: 114%       major: 0        minor: 198      invol: 1        vol: 1

TIME: 0:00.06   CPU: 5%         major: 0        minor: 198      invol: 1        vol: 19
TIME: 0:00.01   CPU: 114%       major: 0        minor: 197      invol: 1        vol: 1

TIME: 0:00.06   CPU: 11%        major: 0        minor: 198      invol: 0        vol: 18
TIME: 0:00.01   CPU: 85%        major: 0        minor: 198      invol: 0        vol: 1

1001
--------------------------------------------------------------------------------

time ctls c_1000.ct:

TIME: 0:00.01   CPU: 0%         major: 0        minor: 126      invol: 1        vol: 6
TIME: 0:00.00   CPU: 0%         major: 0        minor: 125      invol: 1        vol: 2

TIME: 0:00.00   CPU: 0%         major: 0        minor: 126      invol: 0        vol: 6
TIME: 0:00.00   CPU: 0%         major: 0        minor: 125      invol: 1        vol: 2

TIME: 0:00.00   CPU: 0%         major: 0        minor: 126      invol: 0        vol: 6
TIME: 0:00.00   CPU: 0%         major: 0        minor: 125      invol: 1        vol: 2

TIME: 0:00.00   CPU: 0%         major: 0        minor: 126      invol: 0        vol: 6
TIME: 0:00.00   CPU: 0%         major: 0        minor: 126      invol: 1        vol: 2

TIME: 0:00.00   CPU: 200%       major: 0        minor: 125      invol: 1        vol: 6
TIME: 0:00.00   CPU: 0%         major: 0        minor: 126      invol: 1        vol: 2

1000
--------------------------------------------------------------------------------

time ls -l V_1000:

TIME: 0:00.10   CPU: 50%        major: 0        minor: 324      invol: 3        vol: 45
TIME: 0:00.04   CPU: 97%        major: 0        minor: 325      invol: 4        vol: 3

TIME: 0:00.09   CPU: 60%        major: 0        minor: 325      invol: 2        vol: 45
TIME: 0:00.04   CPU: 97%        major: 0        minor: 325      invol: 3        vol: 3

TIME: 0:00.08   CPU: 63%        major: 0        minor: 325      invol: 5        vol: 45
TIME: 0:00.04   CPU: 102%       major: 0        minor: 323      invol: 3        vol: 3

TIME: 0:00.06   CPU: 96%        major: 0        minor: 323      invol: 3        vol: 45
TIME: 0:00.04   CPU: 97%        major: 0        minor: 324      invol: 4        vol: 3

TIME: 0:00.06   CPU: 103%       major: 0        minor: 325      invol: 3        vol: 45
TIME: 0:00.04   CPU: 97%        major: 0        minor: 325      invol: 4        vol: 3

1001
--------------------------------------------------------------------------------

time tar -tvf T_1000.tar:

TIME: 0:00.06   CPU: 26%        major: 0        minor: 208      invol: 2        vol: 25
TIME: 0:00.02   CPU: 109%       major: 0        minor: 209      invol: 1        vol: 1

TIME: 0:00.05   CPU: 48%        major: 0        minor: 209      invol: 5        vol: 26
TIME: 0:00.02   CPU: 90%        major: 0        minor: 208      invol: 1        vol: 1

TIME: 0:00.05   CPU: 43%        major: 0        minor: 208      invol: 2        vol: 21
TIME: 0:00.02   CPU: 95%        major: 0        minor: 208      invol: 1        vol: 1

TIME: 0:00.05   CPU: 42%        major: 0        minor: 209      invol: 3        vol: 31
TIME: 0:00.02   CPU: 109%       major: 0        minor: 207      invol: 2        vol: 1

TIME: 0:00.06   CPU: 28%        major: 0        minor: 209      invol: 4        vol: 19
TIME: 0:00.02   CPU: 90%        major: 0        minor: 209      invol: 1        vol: 1

1001
--------------------------------------------------------------------------------

time ctls -l c_1000.ct:

TIME: 0:00.03   CPU: 12%        major: 0        minor: 156      invol: 3        vol: 10
TIME: 0:00.00   CPU: 133%       major: 0        minor: 158      invol: 4        vol: 4

TIME: 0:00.00   CPU: 133%       major: 0        minor: 157      invol: 3        vol: 4
TIME: 0:00.00   CPU: 133%       major: 0        minor: 157      invol: 4        vol: 4

TIME: 0:00.00   CPU: 133%       major: 0        minor: 157      invol: 4        vol: 4
TIME: 0:00.00   CPU: 133%       major: 0        minor: 157      invol: 3        vol: 4

TIME: 0:00.00   CPU: 133%       major: 0        minor: 157      invol: 4        vol: 4
TIME: 0:00.00   CPU: 133%       major: 0        minor: 158      invol: 4        vol: 4

TIME: 0:00.00   CPU: 0%         major: 0        minor: 157      invol: 3        vol: 4
TIME: 0:00.00   CPU: 0%         major: 0        minor: 157      invol: 4        vol: 4

1000
--------------------------------------------------------------------------------

/dev/md2 on /raid/heinrich type ext3 (rw,noexec,nosuid,nodev)
10000 (1*4 KB) Dateien on ext3
--------------------------------------------------------------------------------

time create_testfiles2.sh V_10000:
TIME: 0:19.19   CPU: 97%        major: 0        minor: 2140521  invol: 17280    vol: 17355

time ctmk c_10000.ct:
TIME: 0:02.76   CPU: 47%        major: 0        minor: 498      invol: 16       vol: 10410

time tar -cvf V_10000:
tar: Removing leading `/' from member names
TIME: 0:58.10   CPU: 2%         major: 0        minor: 276      invol: 33       vol: 10421
--------------------------------------------------------------------------------
```

```
TIME: 0:00.01   CPU: 114%       major: 0        minor: 197      invol: 1        vol: 1

TIME: 0:00.05   CPU: 29%        major: 0        minor: 197      invol: 0        vol: 24
TIME: 0:00.01   CPU: 114%       major: 0        minor: 197      invol: 1        vol: 1

TIME: 0:00.06   CPU: 12%        major: 0        minor: 198      invol: 1        vol: 24
TIME: 0:00.01   CPU: 114%       major: 0        minor: 198      invol: 1        vol: 1

TIME: 0:00.04   CPU: 16%        major: 0        minor: 198      invol: 2        vol: 34
TIME: 0:00.01   CPU: 114%       major: 0        minor: 198      invol: 1        vol: 1

1001
--------------------------------------------------------------------------------

time ctls c_1000.ct:

TIME: 0:00.00   CPU: 0%         major: 0        minor: 125      invol: 1        vol: 7
TIME: 0:00.00   CPU: 400%       major: 0        minor: 126      invol: 1        vol: 2

TIME: 0:00.00   CPU: 0%         major: 0        minor: 126      invol: 0        vol: 6
TIME: 0:00.00   CPU: 0%         major: 0        minor: 126      invol: 0        vol: 2

TIME: 0:00.00   CPU: 200%       major: 0        minor: 125      invol: 0        vol: 6
TIME: 0:00.00   CPU: 400%       major: 0        minor: 125      invol: 1        vol: 2

TIME: 0:00.01   CPU: 0%         major: 0        minor: 126      invol: 1        vol: 6
TIME: 0:00.00   CPU: 400%       major: 0        minor: 126      invol: 1        vol: 2

TIME: 0:00.00   CPU: 0%         major: 0        minor: 124      invol: 1        vol: 6
TIME: 0:00.00   CPU: 400%       major: 0        minor: 126      invol: 1        vol: 2

1000
--------------------------------------------------------------------------------

time ls -l V_1000:

TIME: 0:00.06   CPU: 77%        major: 0        minor: 310      invol: 3        vol: 29
TIME: 0:00.03   CPU: 102%       major: 0        minor: 310      invol: 3        vol: 5

TIME: 0:00.05   CPU: 101%       major: 0        minor: 310      invol: 1        vol: 31
TIME: 0:00.03   CPU: 102%       major: 0        minor: 310      invol: 1        vol: 5

TIME: 0:00.05   CPU: 101%       major: 0        minor: 310      invol: 3        vol: 28
TIME: 0:00.03   CPU: 102%       major: 0        minor: 310      invol: 2        vol: 5

TIME: 0:00.05   CPU: 94%        major: 0        minor: 309      invol: 5        vol: 27
TIME: 0:00.03   CPU: 102%       major: 0        minor: 311      invol: 3        vol: 3

TIME: 0:00.05   CPU: 94%        major: 0        minor: 310      invol: 5        vol: 27
TIME: 0:00.04   CPU: 107%       major: 0        minor: 309      invol: 5        vol: 1

1001
--------------------------------------------------------------------------------

time tar -tvf T_1000.tar:

TIME: 0:00.06   CPU: 39%        major: 0        minor: 208      invol: 4        vol: 21
TIME: 0:00.02   CPU: 109%       major: 0        minor: 208      invol: 1        vol: 1

TIME: 0:00.04   CPU: 57%        major: 0        minor: 208      invol: 1        vol: 24
TIME: 0:00.02   CPU: 109%       major: 0        minor: 208      invol: 2        vol: 1

TIME: 0:00.06   CPU: 51%        major: 0        minor: 207      invol: 3        vol: 18
TIME: 0:00.02   CPU: 109%       major: 0        minor: 208      invol: 1        vol: 1

TIME: 0:00.11   CPU: 27%        major: 0        minor: 208      invol: 3        vol: 19
TIME: 0:00.02   CPU: 114%       major: 0        minor: 208      invol: 1        vol: 1

TIME: 0:00.05   CPU: 50%        major: 0        minor: 210      invol: 2        vol: 18
TIME: 0:00.02   CPU: 109%       major: 0        minor: 208      invol: 1        vol: 1

1001
--------------------------------------------------------------------------------

time ctls -l c_1000.ct:

TIME: 0:00.00   CPU: 44%        major: 0        minor: 158      invol: 5        vol: 7
TIME: 0:00.00   CPU: 133%       major: 0        minor: 158      invol: 5        vol: 2

TIME: 0:00.00   CPU: 133%       major: 0        minor: 157      invol: 3        vol: 6
TIME: 0:00.00   CPU: 133%       major: 0        minor: 157      invol: 1        vol: 6

TIME: 0:00.00   CPU: 133%       major: 0        minor: 157      invol: 1        vol: 6
TIME: 0:00.00   CPU: 133%       major: 0        minor: 158      invol: 5        vol: 2

TIME: 0:00.00   CPU: 133%       major: 0        minor: 158      invol: 3        vol: 6
TIME: 0:00.00   CPU: 133%       major: 0        minor: 158      invol: 1        vol: 6

TIME: 0:00.00   CPU: 133%       major: 0        minor: 159      invol: 1        vol: 6
TIME: 0:00.00   CPU: 133%       major: 0        minor: 158      invol: 5        vol: 2

1000
--------------------------------------------------------------------------------

/dev/md2 on /raid/heinrich type reiserfs (rw,noexec,nosuid,nodev)
10000 (1*4 KB) Dateien on reiserfs
--------------------------------------------------------------------------------

time create_testfiles2.sh V_10000:
TIME: 0:19.89   CPU: 96%        major: 0        minor: 2139571  invol: 17731    vol: 17666

time ctmk c_10000.ct:
TIME: 0:02.31   CPU: 57%        major: 0        minor: 470      invol: 9        vol: 10254

time tar -cvf V_10000:
tar: Removing leading `/' from member names
TIME: 0:02.77   CPU: 46%        major: 0        minor: 276      invol: 50       vol: 10253
--------------------------------------------------------------------------------
```

time ls -l V_10000:

TIME: 0:00.17 CPU: 45% major: 0 minor: 863 invol: 4 vol: 89
TIME: 0:00.08 CPU: 98% major: 0 minor: 862 invol: 2 vol: 2

TIME: 0:00.10 CPU: 80% major: 0 minor: 863 invol: 4 vol: 89
TIME: 0:00.08 CPU: 97% major: 0 minor: 862 invol: 3 vol: 2

TIME: 0:00.09 CPU: 84% major: 0 minor: 861 invol: 2 vol: 89
TIME: 0:00.08 CPU: 97% major: 0 minor: 863 invol: 4 vol: 2

TIME: 0:00.09 CPU: 82% major: 0 minor: 861 invol: 2 vol: 89
TIME: 0:00.08 CPU: 97% major: 0 minor: 861 invol: 2 vol: 2

TIME: 0:00.09 CPU: 101% major: 0 minor: 862 invol: 3 vol: 89
TIME: 0:00.08 CPU: 98% major: 0 minor: 862 invol: 2 vol: 2

10000

---

time tar -tf T_10000.tar:

TIME: 0:00.44 CPU: 30% major: 0 minor: 197 invol: 11 vol: 158
TIME: 0:00.12 CPU: 101% major: 0 minor: 198 invol: 1 vol: 1

TIME: 0:00.56 CPU: 24% major: 0 minor: 198 invol: 8 vol: 131
TIME: 0:00.12 CPU: 98% major: 0 minor: 197 invol: 3 vol: 1

TIME: 0:00.48 CPU: 21% major: 0 minor: 196 invol: 8 vol: 151
TIME: 0:00.12 CPU: 99% major: 0 minor: 197 invol: 1 vol: 1

TIME: 0:00.51 CPU: 27% major: 0 minor: 198 invol: 7 vol: 130
TIME: 0:00.12 CPU: 98% major: 0 minor: 198 invol: 1 vol: 1

TIME: 0:00.51 CPU: 26% major: 0 minor: 198 invol: 6 vol: 121
TIME: 0:00.12 CPU: 100% major: 0 minor: 197 invol: 2 vol: 1

10001

---

time ctls c_10000.ct:

TIME: 0:00.02 CPU: 13% major: 0 minor: 163 invol: 0 vol: 13
TIME: 0:00.00 CPU: 133% major: 0 minor: 162 invol: 0 vol: 2

TIME: 0:00.01 CPU: 26% major: 0 minor: 164 invol: 0 vol: 7
TIME: 0:00.00 CPU: 133% major: 0 minor: 163 invol: 0 vol: 2

TIME: 0:00.00 CPU: 0% major: 0 minor: 163 invol: 0 vol: 7
TIME: 0:00.00 CPU: 133% major: 0 minor: 165 invol: 0 vol: 2

TIME: 0:00.00 CPU: 80% major: 0 minor: 164 invol: 1 vol: 7
TIME: 0:00.00 CPU: 133% major: 0 minor: 165 invol: 0 vol: 2

TIME: 0:00.00 CPU: 0% major: 0 minor: 162 invol: 0 vol: 7
TIME: 0:00.00 CPU: 133% major: 0 minor: 164 invol: 0 vol: 2

10000

---

time ls -l V_10000:

TIME: 0:00.77 CPU: 76% major: 0 minor: 882 invol: 5 vol: 401
TIME: 0:00.40 CPU: 99% major: 0 minor: 881 invol: 4 vol: 3

TIME: 0:00.65 CPU: 90% major: 0 minor: 882 invol: 6 vol: 399
TIME: 0:00.40 CPU: 100% major: 0 minor: 883 invol: 2 vol: 3

TIME: 0:00.62 CPU: 93% major: 0 minor: 881 invol: 8 vol: 399
TIME: 0:00.43 CPU: 100% major: 0 minor: 881 invol: 4 vol: 2

TIME: 0:00.62 CPU: 95% major: 0 minor: 881 invol: 4 vol: 401
TIME: 0:00.42 CPU: 100% major: 0 minor: 881 invol: 11 vol: 2

TIME: 0:00.61 CPU: 93% major: 0 minor: 881 invol: 4 vol: 401
TIME: 0:00.40 CPU: 99% major: 0 minor: 881 invol: 5 vol: 5

10001

---

time tar -tvfT_10000.tar:

TIME: 0:00.50 CPU: 40% major: 0 minor: 208 invol: 14 vol: 161
TIME: 0:00.20 CPU: 100% major: 0 minor: 209 invol: 3 vol: 1

TIME: 0:00.48 CPU: 40% major: 0 minor: 208 invol: 13 vol: 172
TIME: 0:00.20 CPU: 100% major: 0 minor: 208 invol: 1 vol: 1

TIME: 0:00.46 CPU: 51% major: 0 minor: 209 invol: 15 vol: 162
TIME: 0:00.20 CPU: 100% major: 0 minor: 208 invol: 1 vol: 1

TIME: 0:00.48 CPU: 41% major: 0 minor: 208 invol: 10 vol: 159
TIME: 0:00.20 CPU: 100% major: 0 minor: 208 invol: 2 vol: 1

TIME: 0:00.50 CPU: 38% major: 0 minor: 208 invol: 10 vol: 156
TIME: 0:00.20 CPU: 100% major: 0 minor: 208 invol: 1 vol: 1

10001

---

time ctls -l c_10000.ct:

TIME: 0:00.05 CPU: 40% major: 0 minor: 231 invol: 4 vol: 11
TIME: 0:00.01 CPU: 105% major: 0 minor: 231 invol: 3 vol: 4

TIME: 0:00.01 CPU: 84% major: 0 minor: 231 invol: 3 vol: 6
TIME: 0:00.02 CPU: 100% major: 0 minor: 232 invol: 4 vol: 4

TIME: 0:00.02 CPU: 100% major: 0 minor: 232 invol: 1 vol: 6

time ls -l V_10000:

TIME: 0:00.09 CPU: 62% major: 0 minor: 717 invol: 5 vol: 10
TIME: 0:00.04 CPU: 97% major: 0 minor: 716 invol: 4 vol: 3

TIME: 0:00.05 CPU: 77% major: 0 minor: 717 invol: 2 vol: 13
TIME: 0:00.04 CPU: 97% major: 0 minor: 717 invol: 1 vol: 5

TIME: 0:00.05 CPU: 75% major: 0 minor: 716 invol: 3 vol: 10
TIME: 0:00.04 CPU: 97% major: 0 minor: 715 invol: 3 vol: 3

TIME: 0:00.06 CPU: 73% major: 0 minor: 718 invol: 1 vol: 15
TIME: 0:00.04 CPU: 106% major: 0 minor: 715 invol: 6 vol: 1

TIME: 0:00.05 CPU: 82% major: 0 minor: 716 invol: 4 vol: 11
TIME: 0:00.04 CPU: 97% major: 0 minor: 715 invol: 5 vol: 1

10000

---

time tar -tf T_10000.tar:

TIME: 0:00.43 CPU: 35% major: 0 minor: 198 invol: 26 vol: 177
TIME: 0:00.12 CPU: 105% major: 0 minor: 197 invol: 1 vol: 1

TIME: 0:00.43 CPU: 34% major: 0 minor: 197 invol: 13 vol: 183
TIME: 0:00.12 CPU: 98% major: 0 minor: 198 invol: 1 vol: 1

TIME: 0:00.48 CPU: 38% major: 0 minor: 197 invol: 19 vol: 148
TIME: 0:00.12 CPU: 99% major: 0 minor: 196 invol: 1 vol: 1

TIME: 0:00.45 CPU: 30% major: 0 minor: 197 invol: 14 vol: 164
TIME: 0:00.12 CPU: 102% major: 0 minor: 199 invol: 2 vol: 1

TIME: 0:00.49 CPU: 34% major: 0 minor: 197 invol: 18 vol: 161
TIME: 0:00.12 CPU: 99% major: 0 minor: 197 invol: 0 vol: 1

10001

---

time ctls c_10000.ct:

TIME: 0:00.05 CPU: 7% major: 0 minor: 162 invol: 2 vol: 9
TIME: 0:00.00 CPU: 0% major: 0 minor: 163 invol: 1 vol: 2

TIME: 0:00.00 CPU: 57% major: 0 minor: 164 invol: 1 vol: 6
TIME: 0:00.00 CPU: 0% major: 0 minor: 164 invol: 0 vol: 2

TIME: 0:00.01 CPU: 0% major: 0 minor: 163 invol: 1 vol: 6
TIME: 0:00.00 CPU: 0% major: 0 minor: 164 invol: 1 vol: 2

TIME: 0:00.01 CPU: 0% major: 0 minor: 164 invol: 1 vol: 6
TIME: 0:00.00 CPU: 0% major: 0 minor: 164 invol: 1 vol: 2

TIME: 0:00.00 CPU: 80% major: 0 minor: 163 invol: 1 vol: 6
TIME: 0:00.00 CPU: 0% major: 0 minor: 164 invol: 1 vol: 2

10000

---

time ls -l V_10000:

TIME: 0:00.62 CPU: 86% major: 0 minor: 735 invol: 5 vol: 247
TIME: 0:00.37 CPU: 99% major: 0 minor: 734 invol: 9 vol: 1

TIME: 0:00.62 CPU: 89% major: 0 minor: 735 invol: 3 vol: 249
TIME: 0:00.37 CPU: 99% major: 0 minor: 735 invol: 2 vol: 5

TIME: 0:00.61 CPU: 84% major: 0 minor: 735 invol: 5 vol: 249
TIME: 0:00.37 CPU: 100% major: 0 minor: 735 invol: 0 vol: 5

TIME: 0:00.61 CPU: 89% major: 0 minor: 737 invol: 5 vol: 246
TIME: 0:00.39 CPU: 99% major: 0 minor: 735 invol: 4 vol: 5

TIME: 0:00.66 CPU: 76% major: 0 minor: 736 invol: 4 vol: 245
TIME: 0:00.37 CPU: 99% major: 0 minor: 736 invol: 9 vol: 1

10001

---

time tar -tvfT_10000.tar:

TIME: 0:00.46 CPU: 59% major: 0 minor: 207 invol: 17 vol: 159
TIME: 0:00.20 CPU: 99% major: 0 minor: 209 invol: 1 vol: 1

TIME: 0:00.46 CPU: 56% major: 0 minor: 208 invol: 12 vol: 171
TIME: 0:00.20 CPU: 99% major: 0 minor: 208 invol: 3 vol: 1

TIME: 0:00.49 CPU: 54% major: 0 minor: 208 invol: 18 vol: 164
TIME: 0:00.20 CPU: 99% major: 0 minor: 209 invol: 1 vol: 1

TIME: 0:00.46 CPU: 55% major: 0 minor: 209 invol: 15 vol: 165
TIME: 0:00.20 CPU: 100% major: 0 minor: 208 invol: 0 vol: 1

TIME: 0:00.57 CPU: 41% major: 0 minor: 208 invol: 14 vol: 170
TIME: 0:00.20 CPU: 99% major: 0 minor: 209 invol: 1 vol: 1

10001

---

time ctls -l c_10000.ct:

TIME: 0:00.03 CPU: 52% major: 0 minor: 231 invol: 1 vol: 12
TIME: 0:00.02 CPU: 100% major: 0 minor: 231 invol: 3 vol: 4

TIME: 0:00.02 CPU: 100% major: 0 minor: 233 invol: 4 vol: 4
TIME: 0:00.01 CPU: 105% major: 0 minor: 232 invol: 1 vol: 6

TIME: 0:00.01 CPU: 105% major: 0 minor: 231 invol: 3 vol: 4

TIME: 0:00.02 CPU: 100% major: 0 minor: 231 invol: 5 vol: 2	TIME: 0:00.01 CPU: 88% major: 0 minor: 232 invol: 4 vol: 4
TIME: 0:00.01 CPU: 84% major: 0 minor: 231 invol: 5 vol: 2	TIME: 0:00.02 CPU: 114% major: 0 minor: 232 invol: 1 vol: 6
TIME: 0:00.01 CPU: 84% major: 0 minor: 231 invol: 3 vol: 6	TIME: 0:00.01 CPU: 111% major: 0 minor: 232 invol: 3 vol: 4
TIME: 0:00.01 CPU: 84% major: 0 minor: 230 invol: 3 vol: 4	TIME: 0:00.02 CPU: 100% major: 0 minor: 230 invol: 4 vol: 4
TIME: 0:00.01 CPU: 84% major: 0 minor: 232 invol: 3 vol: 6	TIME: 0:00.01 CPU: 105% major: 0 minor: 232 invol: 4 vol: 2
10000	10000
---	---
/dev/md2 on /raid/heinrich type ext3 (rw,noexec,nosuid,nodev)	/dev/md2 on /raid/heinrich type reiserfs (rw,noexec,nosuid,nodev)
100000 (1*4 KB) Dateien on ext3	100000 (1*4 KB) Dateien on reiserfs
---	---
time create_testfiles2.sh V_100000:	time create_testfiles2.sh V_100000:
TIME: 3:15.09 CPU: 94% major: 0 minor: 21389197 invol: 173951 vol: 174325	TIME: 3:20.50 CPU: 99% major: 0 minor: 21388216 invol: 178758 vol: 177544
time ctmk c_100000.ct:	time ctmk c_100000.ct:
TIME: 0:41.47 CPU: 29% major: 0 minor: 3697 invol: 1441 vol: 104027	TIME: 0:28.96 CPU: 48% major: 0 minor: 3450 invol: 984 vol: 102484
time tar -cvf V_100000:	time tar -cvf V_100000:
tar: Removing leading `/' from member names	tar: Removing leading `/' from member names
TIME: 16:56.29 CPU: 1% major: 0 minor: 649 invol: 908 vol: 104254	TIME: 0:28.97 CPU: 50% major: 0 minor: 650 invol: 800 vol: 102476
---	---
time ls -l V_100000:	time ls -l V_100000:
TIME: 0:01.35 CPU: 80% major: 0 minor: 6664 invol: 9 vol: 884	TIME: 0:00.62 CPU: 81% major: 0 minor: 5199 invol: 10 vol: 72
TIME: 0:01.07 CPU: 100% major: 0 minor: 6663 invol: 6 vol: 2	TIME: 0:00.52 CPU: 99% major: 0 minor: 5199 invol: 4 vol: 2
TIME: 0:01.28 CPU: 83% major: 0 minor: 6663 invol: 7 vol: 884	TIME: 0:00.55 CPU: 93% major: 0 minor: 5199 invol: 7 vol: 69
TIME: 0:01.07 CPU: 100% major: 0 minor: 6664 invol: 10 vol: 1	TIME: 0:00.51 CPU: 100% major: 0 minor: 5199 invol: 3 vol: 4
TIME: 0:01.16 CPU: 92% major: 0 minor: 6663 invol: 9 vol: 882	TIME: 0:00.57 CPU: 93% major: 0 minor: 5198 invol: 4 vol: 70
TIME: 0:01.06 CPU: 100% major: 0 minor: 6664 invol: 7 vol: 3	TIME: 0:00.51 CPU: 99% major: 0 minor: 5198 invol: 1 vol: 3
TIME: 0:01.17 CPU: 91% major: 0 minor: 6664 invol: 8 vol: 883	TIME: 0:00.58 CPU: 91% major: 0 minor: 5198 invol: 11 vol: 73
TIME: 0:01.06 CPU: 100% major: 0 minor: 6663 invol: 4 vol: 5	TIME: 0:00.51 CPU: 100% major: 0 minor: 5198 invol: 6 vol: 5
TIME: 0:01.16 CPU: 92% major: 0 minor: 6663 invol: 9 vol: 884	TIME: 0:00.56 CPU: 96% major: 0 minor: 5198 invol: 7 vol: 67
TIME: 0:01.06 CPU: 100% major: 0 minor: 6664 invol: 10 vol: 3	TIME: 0:00.51 CPU: 100% major: 0 minor: 5200 invol: 8 vol: 2
100000	100000
---	---
time tar -tf T_100000.tar:	time tar -tf T_100000.tar:
TIME: 0:04.61 CPU: 28% major: 0 minor: 197 invol: 97 vol: 1427	TIME: 0:04.34 CPU: 31% major: 0 minor: 197 invol: 211 vol: 1640
TIME: 0:01.25 CPU: 100% major: 0 minor: 196 invol: 10 vol: 1	TIME: 0:01.24 CPU: 100% major: 0 minor: 197 invol: 11 vol: 1
TIME: 0:04.39 CPU: 23% major: 0 minor: 197 invol: 120 vol: 1556	TIME: 0:04.46 CPU: 30% major: 0 minor: 198 invol: 214 vol: 1624
TIME: 0:01.24 CPU: 100% major: 0 minor: 197 invol: 6 vol: 5	TIME: 0:01.24 CPU: 100% major: 0 minor: 198 invol: 12 vol: 1
TIME: 0:04.47 CPU: 24% major: 0 minor: 198 invol: 116 vol: 1443	TIME: 0:04.45 CPU: 31% major: 0 minor: 197 invol: 207 vol: 1571
TIME: 0:01.23 CPU: 100% major: 0 minor: 197 invol: 4 vol: 1	TIME: 0:01.24 CPU: 100% major: 0 minor: 198 invol: 10 vol: 1
TIME: 0:04.64 CPU: 27% major: 0 minor: 197 invol: 65 vol: 1417	TIME: 0:04.38 CPU: 31% major: 0 minor: 197 invol: 185 vol: 1596
TIME: 0:01.25 CPU: 100% major: 0 minor: 196 invol: 6 vol: 1	TIME: 0:01.23 CPU: 100% major: 0 minor: 197 invol: 12 vol: 1
TIME: 0:04.57 CPU: 24% major: 0 minor: 197 invol: 84 vol: 1372	TIME: 0:04.47 CPU: 31% major: 0 minor: 197 invol: 211 vol: 1541
TIME: 0:01.25 CPU: 99% major: 0 minor: 197 invol: 4 vol: 1	TIME: 0:01.24 CPU: 100% major: 0 minor: 198 invol: 10 vol: 1
100001	100001
---	---
time ctls c_100000.ct:	time ctls c_100000.ct:
TIME: 0:00.08 CPU: 28% major: 0 minor: 537 invol: 2 vol: 17	TIME: 0:00.07 CPU: 30% major: 0 minor: 537 invol: 3 vol: 20
TIME: 0:00.02 CPU: 114% major: 0 minor: 538 invol: 1 vol: 2	TIME: 0:00.02 CPU: 114% major: 0 minor: 537 invol: 1 vol: 2
TIME: 0:00.06 CPU: 32% major: 0 minor: 538 invol: 2 vol: 13	TIME: 0:00.05 CPU: 45% major: 0 minor: 538 invol: 1 vol: 12
TIME: 0:00.02 CPU: 133% major: 0 minor: 537 invol: 1 vol: 2	TIME: 0:00.02 CPU: 109% major: 0 minor: 538 invol: 1 vol: 2
TIME: 0:00.04 CPU: 40% major: 0 minor: 536 invol: 1 vol: 14	TIME: 0:00.04 CPU: 57% major: 0 minor: 537 invol: 2 vol: 13
TIME: 0:00.02 CPU: 114% major: 0 minor: 537 invol: 1 vol: 2	TIME: 0:00.02 CPU: 109% major: 0 minor: 538 invol: 1 vol: 2
TIME: 0:00.04 CPU: 50% major: 0 minor: 537 invol: 2 vol: 14	TIME: 0:00.06 CPU: 45% major: 0 minor: 538 invol: 1 vol: 21
TIME: 0:00.02 CPU: 100% major: 0 minor: 537 invol: 0 vol: 2	TIME: 0:00.02 CPU: 100% major: 0 minor: 537 invol: 1 vol: 2
TIME: 0:00.04 CPU: 40% major: 0 minor: 537 invol: 2 vol: 13	TIME: 0:00.05 CPU: 40% major: 0 minor: 537 invol: 1 vol: 13
TIME: 0:00.02 CPU: 95% major: 0 minor: 538 invol: 1 vol: 2	TIME: 0:00.02 CPU: 95% major: 0 minor: 538 invol: 1 vol: 2
100000	100000
---	---
time ls -l V_100000:	time ls -l V_100000:
TIME: 0:22.81 CPU: 27% major: 0 minor: 6681 invol: 21 vol: 4011	TIME: 0:06.32 CPU: 86% major: 0 minor: 5218 invol: 36 vol: 2464
TIME: 0:04.33 CPU: 99% major: 0 minor: 6682 invol: 24 vol: 5	TIME: 0:03.96 CPU: 100% major: 0 minor: 5218 invol: 32 vol: 2
TIME: 0:23.43 CPU: 26% major: 0 minor: 6682 invol: 32 vol: 4012	TIME: 0:06.56 CPU: 87% major: 0 minor: 5217 invol: 41 vol: 2469
TIME: 0:04.35 CPU: 99% major: 0 minor: 6682 invol: 26 vol: 6	TIME: 0:03.85 CPU: 100% major: 0 minor: 5217 invol: 32 vol: 7
TIME: 0:23.24 CPU: 26% major: 0 minor: 6682 invol: 23 vol: 4010	TIME: 0:06.34 CPU: 86% major: 0 minor: 5218 invol: 33 vol: 2467
TIME: 0:04.36 CPU: 100% major: 0 minor: 6683 invol: 23 vol: 7	TIME: 0:03.89 CPU: 99% major: 0 minor: 5217 invol: 28 vol: 6
TIME: 0:23.41 CPU: 26% major: 0 minor: 6682 invol: 26 vol: 4014	TIME: 0:06.47 CPU: 86% major: 0 minor: 5218 invol: 22 vol: 2462
TIME: 0:04.41 CPU: 100% major: 0 minor: 6683 invol: 31 vol: 4	TIME: 0:03.83 CPU: 99% major: 0 minor: 5218 invol: 22 vol: 10
TIME: 0:22.61 CPU: 27% major: 0 minor: 6682 invol: 15 vol: 4014	TIME: 0:06.61 CPU: 80% major: 0 minor: 5217 invol: 17 vol: 2465
TIME: 0:04.39 CPU: 99% major: 0 minor: 6682 invol: 15 vol: 4	TIME: 0:03.87 CPU: 99% major: 0 minor: 5217 invol: 20 vol: 2
100001	100001
---	---
time tar -tvf T_100000.tar:	time tar -tvf T_100000.tar:
TIME: 0:04.60 CPU: 42% major: 0 minor: 209 invol: 105 vol: 1525	TIME: 0:04.54 CPU: 53% major: 0 minor: 209 invol: 211 vol: 1626

TIME: 0:02.02 CPU: 99% major: 0 minor: 209 invol: 5 vol: 1	TIME: 0:01.98 CPU: 99% major: 0 minor: 208 invol: 17 vol: 2
TIME: 0:04.32 CPU: 44% major: 0 minor: 209 invol: 147 vol: 1650	TIME: 0:04.48 CPU: 51% major: 0 minor: 210 invol: 245 vol: 1624
TIME: 0:01.97 CPU: 99% major: 0 minor: 208 invol: 3 vol: 1	TIME: 0:02.01 CPU: 99% major: 0 minor: 209 invol: 18 vol: 2
TIME: 0:04.58 CPU: 42% major: 0 minor: 209 invol: 111 vol: 1531	TIME: 0:04.53 CPU: 53% major: 0 minor: 208 invol: 223 vol: 1571
TIME: 0:02.00 CPU: 99% major: 0 minor: 209 invol: 13 vol: 5	TIME: 0:02.02 CPU: 100% major: 0 minor: 208 invol: 16 vol: 1
TIME: 0:04.64 CPU: 41% major: 0 minor: 209 invol: 100 vol: 1531	TIME: 0:04.55 CPU: 51% major: 0 minor: 208 invol: 255 vol: 1576
TIME: 0:02.00 CPU: 99% major: 0 minor: 209 invol: 5 vol: 1	TIME: 0:02.00 CPU: 99% major: 0 minor: 209 invol: 16 vol: 5
TIME: 0:04.42 CPU: 43% major: 0 minor: 209 invol: 153 vol: 1693	TIME: 0:04.57 CPU: 51% major: 0 minor: 208 invol: 284 vol: 1581
TIME: 0:02.00 CPU: 100% major: 0 minor: 208 invol: 21 vol: 1	TIME: 0:01.98 CPU: 100% major: 0 minor: 209 invol: 12 vol: 1
100001	100001
---	---
time ctls -l c_100000.ct:	time ctls -l c_100000.ct:
TIME: 0:00.24 CPU: 73% major: 0 minor: 957 invol: 8 vol: 26	TIME: 0:00.24 CPU: 75% major: 0 minor: 956 invol: 5 vol: 27
TIME: 0:00.17 CPU: 100% major: 0 minor: 956 invol: 5 vol: 2	TIME: 0:00.18 CPU: 101% major: 0 minor: 956 invol: 4 vol: 4
TIME: 0:00.17 CPU: 98% major: 0 minor: 957 invol: 2 vol: 4	TIME: 0:00.18 CPU: 100% major: 0 minor: 955 invol: 1 vol: 6
TIME: 0:00.17 CPU: 98% major: 0 minor: 956 invol: 3 vol: 4	TIME: 0:00.18 CPU: 100% major: 0 minor: 957 invol: 6 vol: 4
TIME: 0:00.16 CPU: 100% major: 0 minor: 958 invol: 4 vol: 4	TIME: 0:00.19 CPU: 98% major: 0 minor: 956 invol: 3 vol: 4
TIME: 0:00.17 CPU: 100% major: 0 minor: 956 invol: 3 vol: 6	TIME: 0:00.18 CPU: 100% major: 0 minor: 957 invol: 1 vol: 6
TIME: 0:00.17 CPU: 99% major: 0 minor: 956 invol: 8 vol: 2	TIME: 0:00.17 CPU: 98% major: 0 minor: 957 invol: 5 vol: 4
TIME: 0:00.18 CPU: 100% major: 0 minor: 955 invol: 5 vol: 2	TIME: 0:00.18 CPU: 99% major: 0 minor: 957 invol: 3 vol: 4
TIME: 0:00.17 CPU: 99% major: 0 minor: 956 invol: 4 vol: 4	TIME: 0:00.18 CPU: 100% major: 0 minor: 955 invol: 6 vol: 2
TIME: 0:00.18 CPU: 99% major: 0 minor: 955 invol: 3 vol: 4	TIME: 0:00.17 CPU: 100% major: 0 minor: 957 invol: 5 vol: 4
100000	100000
---	---
/dev/md2 on /raid/heinrich type ext3 (rw,noexec,nosuid,nodev) 1000000 (1*4 KB) Dateien on ext3	/dev/md2 on /raid/heinrich type reiserfs (rw,noexec,nosuid,nodev) 1000000 (1*4 KB) Dateien on reiserfs
---	---
time create_testfiles2.sh V_1000000: TIME: 35:08.76 CPU: 79% major: 0 minor: 213874847 invol: 1776082 vol: 1774846	time create_testfiles2.sh V_1000000: TIME: 33:37.36 CPU: 105% major: 0 minor: 213885005 invol: 1851371 vol: 1824333
time ctmk c_1000000.ct: TIME: 11:05.70 CPU: 24% major: 0 minor: 35571 invol: 22332 vol: 1070766	time ctmk c_1000000.ct: TIME: 6:01.18 CPU: 48% major: 0 minor: 33114 invol: 11351 vol: 1054860
time tar -cvfV_1000000: tar: Removing leading `/' from member names TIME: 3:29:26 CPU: 1% major: 4 minor: 4380 invol: 16593 vol: 1065196	time tar -cvfV_1000000: tar: Removing leading `/' from member names TIME: 5:33.82 CPU: 45% major: 4 minor: 4381 invol: 12143 vol: 1031410
---	---
time ls -l V_1000000:	time ls -l V_1000000:
TIME: 0:52.35 CPU: 26% major: 3 minor: 64667 invol: 107 vol: 8440	TIME: 0:06.58 CPU: 93% major: 4 minor: 50019 invol: 45 vol: 675
TIME: 0:13.47 CPU: 100% major: 0 minor: 64669 invol: 48 vol: 8	TIME: 0:06.02 CPU: 99% major: 0 minor: 50022 invol: 24 vol: 3
TIME: 0:49.72 CPU: 27% major: 0 minor: 64670 invol: 79 vol: 8430	TIME: 0:06.38 CPU: 95% major: 0 minor: 50023 invol: 44 vol: 657
TIME: 0:13.46 CPU: 99% major: 0 minor: 64670 invol: 60 vol: 8	TIME: 0:06.03 CPU: 100% major: 0 minor: 50024 invol: 28 vol: 6
TIME: 0:50.35 CPU: 27% major: 0 minor: 64670 invol: 83 vol: 8434	TIME: 0:06.69 CPU: 95% major: 0 minor: 50024 invol: 36 vol: 657
TIME: 0:13.44 CPU: 99% major: 0 minor: 64670 invol: 51 vol: 10	TIME: 0:06.03 CPU: 100% major: 0 minor: 50024 invol: 25 vol: 2
TIME: 0:50.01 CPU: 27% major: 0 minor: 64670 invol: 81 vol: 8432	TIME: 0:06.39 CPU: 95% major: 0 minor: 50023 invol: 41 vol: 655
TIME: 0:13.45 CPU: 99% major: 0 minor: 64671 invol: 53 vol: 8	TIME: 0:06.03 CPU: 99% major: 0 minor: 50023 invol: 25 vol: 3
TIME: 0:50.41 CPU: 26% major: 0 minor: 64671 invol: 74 vol: 8430	TIME: 0:06.39 CPU: 95% major: 0 minor: 50023 invol: 25 vol: 657
TIME: 0:13.50 CPU: 100% major: 0 minor: 64669 invol: 59 vol: 6	TIME: 0:06.06 CPU: 100% major: 0 minor: 50021 invol: 36 vol: 2
1000000	1000000
---	---
time tar -tf T_1000000.tar:	time tar -tf T_1000000.tar:
TIME: 0:52.31 CPU: 24% major: 2 minor: 195 invol: 2555 vol: 15897	TIME: 0:43.70 CPU: 34% major: 2 minor: 196 invol: 3487 vol: 16046
TIME: 0:49.11 CPU: 26% major: 1 minor: 196 invol: 3139 vol: 15310	TIME: 0:43.74 CPU: 35% major: 1 minor: 196 invol: 3866 vol: 16328
TIME: 0:53.86 CPU: 23% major: 3 minor: 194 invol: 2670 vol: 15511	TIME: 0:43.76 CPU: 34% major: 3 minor: 194 invol: 3546 vol: 16009
TIME: 0:48.98 CPU: 26% major: 3 minor: 194 invol: 3078 vol: 15452	TIME: 0:43.94 CPU: 35% major: 3 minor: 194 invol: 3873 vol: 16223
TIME: 0:48.21 CPU: 27% major: 3 minor: 194 invol: 2541 vol: 15689	TIME: 0:43.92 CPU: 34% major: 3 minor: 194 invol: 3502 vol: 16016
TIME: 0:52.17 CPU: 24% major: 3 minor: 194 invol: 3188 vol: 15559	TIME: 0:44.12 CPU: 35% major: 3 minor: 195 invol: 3866 vol: 16428
TIME: 0:53.23 CPU: 24% major: 3 minor: 194 invol: 2674 vol: 15278	TIME: 0:43.60 CPU: 34% major: 3 minor: 195 invol: 3460 vol: 16316
TIME: 0:48.94 CPU: 26% major: 3 minor: 194 invol: 3010 vol: 15790	TIME: 0:44.10 CPU: 35% major: 3 minor: 195 invol: 3980 vol: 16343
TIME: 0:50.11 CPU: 26% major: 3 minor: 194 invol: 2765 vol: 15584	TIME: 0:43.74 CPU: 34% major: 3 minor: 195 invol: 3505 vol: 16055
TIME: 0:49.44 CPU: 26% major: 3 minor: 195 invol: 3125 vol: 15822	TIME: 0:43.77 CPU: 35% major: 3 minor: 194 invol: 3950 vol: 16408
1000001	1000001
---	---
time ctls c_1000000.ct:	time ctls c_1000000.ct:
TIME: 0:00.39 CPU: 53% major: 0 minor: 4273 invol: 3 vol: 101	TIME: 0:00.36 CPU: 66% major: 0 minor: 4272 invol: 16 vol: 96
TIME: 0:00.20 CPU: 101% major: 0 minor: 4274 invol: 1 vol: 2	TIME: 0:00.19 CPU: 100% major: 0 minor: 4272 invol: 4 vol: 2
TIME: 0:00.37 CPU: 49% major: 0 minor: 4272 invol: 4 vol: 76	TIME: 0:00.38 CPU: 67% major: 0 minor: 4272 invol: 14 vol: 71
TIME: 0:00.19 CPU: 101% major: 0 minor: 4273 invol: 1 vol: 2	TIME: 0:00.19 CPU: 99% major: 0 minor: 4272 invol: 3 vol: 2
TIME: 0:00.38 CPU: 57% major: 0 minor: 4272 invol: 3 vol: 82	TIME: 0:00.35 CPU: 62% major: 0 minor: 4272 invol: 14 vol: 74
TIME: 0:00.19 CPU: 101% major: 0 minor: 4274 invol: 4 vol: 2	TIME: 0:00.20 CPU: 100% major: 0 minor: 4273 invol: 2 vol: 2
TIME: 0:00.39 CPU: 53% major: 0 minor: 4273 invol: 4 vol: 71	TIME: 0:00.35 CPU: 59% major: 0 minor: 4272 invol: 14 vol: 80
TIME: 0:00.20 CPU: 100% major: 0 minor: 4274 invol: 1 vol: 2	TIME: 0:00.19 CPU: 99% major: 0 minor: 4272 invol: 2 vol: 2
TIME: 0:00.38 CPU: 54% major: 0 minor: 4272 invol: 2 vol: 81	TIME: 0:00.37 CPU: 57% major: 0 minor: 4273 invol: 13 vol: 61
TIME: 0:00.21 CPU: 100% major: 0 minor: 4273 invol: 1 vol: 3	TIME: 0:00.20 CPU: 99% major: 0 minor: 4271 invol: 1 vol: 3

1000000

---

time ls -l V_1000000:

TIME: 7:12.62 CPU: 16% major: 6 minor: 64684 invol: 367 vol: 39778
TIME: 2:41.41 CPU: 35% major: 0 minor: 64689 invol: 220 vol: 13704

TIME: 7:10.95 CPU: 16% major: 0 minor: 64690 invol: 327 vol: 39767
TIME: 2:34.27 CPU: 37% major: 3 minor: 64686 invol: 233 vol: 13053

TIME: 7:13.89 CPU: 16% major: 0 minor: 64689 invol: 359 vol: 39770
TIME: 2:38.99 CPU: 36% major: 0 minor: 64689 invol: 257 vol: 13396

TIME: 7:10.10 CPU: 16% major: 0 minor: 64689 invol: 347 vol: 39771
TIME: 2:42.12 CPU: 37% major: 3 minor: 64687 invol: 292 vol: 13564

TIME: 7:12.96 CPU: 16% major: 0 minor: 64691 invol: 340 vol: 39773
TIME: 2:32.60 CPU: 38% major: 2 minor: 64687 invol: 252 vol: 12869

1000001

---

time tar -tvf T_1000000.tar:

TIME: 0:47.57 CPU: 42% major: 2 minor: 206 invol: 2914 vol: 17159
TIME: 0:52.05 CPU: 37% major: 1 minor: 208 invol: 3160 vol: 17136

TIME: 0:49.39 CPU: 40% major: 3 minor: 206 invol: 2867 vol: 17118
TIME: 0:51.50 CPU: 38% major: 3 minor: 205 invol: 3193 vol: 17138

TIME: 0:46.90 CPU: 41% major: 3 minor: 205 invol: 2961 vol: 17001
TIME: 0:51.53 CPU: 38% major: 3 minor: 205 invol: 3172 vol: 17051

TIME: 0:50.09 CPU: 39% major: 3 minor: 205 invol: 2699 vol: 17169
TIME: 0:46.60 CPU: 42% major: 3 minor: 206 invol: 3222 vol: 17241

TIME: 0:46.00 CPU: 43% major: 3 minor: 207 invol: 2897 vol: 17338
TIME: 0:46.02 CPU: 43% major: 3 minor: 205 invol: 3300 vol: 17346

1000001

---

time ctls -l c_1000000.ct:

TIME: 0:02.07 CPU: 84% major: 0 minor: 8206 invol: 12 vol: 172
TIME: 0:01.73 CPU: 99% major: 0 minor: 8207 invol: 14 vol: 4

TIME: 0:01.76 CPU: 99% major: 0 minor: 8208 invol: 9 vol: 5
TIME: 0:01.82 CPU: 99% major: 0 minor: 8207 invol: 13 vol: 4

TIME: 0:01.79 CPU: 99% major: 0 minor: 8207 invol: 9 vol: 5
TIME: 0:01.69 CPU: 100% major: 0 minor: 8207 invol: 10 vol: 6

TIME: 0:01.89 CPU: 99% major: 0 minor: 8207 invol: 19 vol: 3
TIME: 0:01.72 CPU: 99% major: 0 minor: 8208 invol: 11 vol: 6

TIME: 0:01.77 CPU: 100% major: 0 minor: 8206 invol: 7 vol: 2
TIME: 0:01.74 CPU: 100% major: 0 minor: 8208 invol: 14 vol: 6

1000000

/dev/md2 on /raid/heinrich type ext3 (rw,noexec,nosuid,nodev)
100 (1*4 KB) Dateien on ext3

---

time create_testfiles2.sh V_100:
TIME: 0:00.27 CPU: 72% major: 3 minor: 22100 invol: 177 vol: 196

time ctmk c_100.ct:
TIME: 0:00.02 CPU: 32% major: 0 minor: 135 invol: 2 vol: 117

---

avtest V_100:

TIME: 0:00.15 CPU: 2% major: 0 minor: 132 invol: 1 vol: 113
TIME: 0:00.00 CPU: 100% major: 0 minor: 130 invol: 0 vol: 2

TIME: 0:00.18 CPU: 2% major: 0 minor: 131 invol: 1 vol: 108
TIME: 0:00.00 CPU: 100% major: 0 minor: 131 invol: 0 vol: 2

TIME: 0:00.14 CPU: 5% major: 0 minor: 132 invol: 0 vol: 108
TIME: 0:00.00 CPU: 100% major: 0 minor: 132 invol: 0 vol: 2

TIME: 0:00.15 CPU: 2% major: 0 minor: 131 invol: 0 vol: 108
TIME: 0:00.00 CPU: 100% major: 0 minor: 131 invol: 1 vol: 2

TIME: 0:00.18 CPU: 4% major: 0 minor: 131 invol: 0 vol: 108
TIME: 0:00.00 CPU: 100% major: 0 minor: 131 invol: 0 vol: 2

100

---

actest c_100.ct:

TIME: 0:00.02 CPU: 28% major: 0 minor: 135 invol: 0 vol: 15
TIME: 0:00.00 CPU: 0% major: 0 minor: 134 invol: 0 vol: 2

TIME: 0:00.01 CPU: 22% major: 0 minor: 137 invol: 0 vol: 11
TIME: 0:00.00 CPU: 0% major: 0 minor: 136 invol: 0 vol: 2

TIME: 0:00.01 CPU: 23% major: 0 minor: 134 invol: 0 vol: 10
TIME: 0:00.00 CPU: 0% major: 0 minor: 135 invol: 0 vol: 2

TIME: 0:00.01 CPU: 33% major: 0 minor: 136 invol: 0 vol: 11
TIME: 0:00.00 CPU: 0% major: 0 minor: 134 invol: 0 vol: 2

1000000

---

time ls -l V_1000000:

TIME: 1:20.83 CPU: 76% major: 4 minor: 50037 invol: 299 vol: 24588
TIME: 0:46.83 CPU: 99% major: 0 minor: 50041 invol: 202 vol: 4

TIME: 1:58.76 CPU: 83% major: 0 minor: 50042 invol: 324 vol: 24617
TIME: 0:44.11 CPU: 99% major: 0 minor: 50040 invol: 251 vol: 23

TIME: 1:22.07 CPU: 77% major: 0 minor: 50041 invol: 197 vol: 24581
TIME: 0:45.12 CPU: 99% major: 0 minor: 50041 invol: 256 vol: 22

TIME: 1:21.40 CPU: 77% major: 0 minor: 50041 invol: 190 vol: 24591
TIME: 0:45.56 CPU: 99% major: 0 minor: 50041 invol: 250 vol: 22

TIME: 1:23.91 CPU: 77% major: 0 minor: 50041 invol: 123 vol: 24627
TIME: 0:44.55 CPU: 99% major: 0 minor: 50042 invol: 237 vol: 21

1000001

---

time tar -tvf T_1000000.tar:

TIME: 0:45.03 CPU: 54% major: 1 minor: 206 invol: 4032 vol: 15091
TIME: 0:45.45 CPU: 53% major: 1 minor: 207 invol: 4635 vol: 14924

TIME: 0:45.58 CPU: 55% major: 3 minor: 206 invol: 4281 vol: 14658
TIME: 0:45.54 CPU: 54% major: 3 minor: 205 invol: 4632 vol: 14867

TIME: 0:45.42 CPU: 54% major: 3 minor: 205 invol: 4268 vol: 14870
TIME: 0:45.80 CPU: 55% major: 3 minor: 206 invol: 4699 vol: 14616

TIME: 0:45.31 CPU: 52% major: 3 minor: 205 invol: 4051 vol: 15174
TIME: 0:45.60 CPU: 54% major: 3 minor: 204 invol: 4691 vol: 14874

TIME: 0:45.40 CPU: 56% major: 3 minor: 206 invol: 4237 vol: 14847
TIME: 0:45.25 CPU: 54% major: 3 minor: 205 invol: 4577 vol: 15102

1000001

---

time ctls -l c_1000000.ct:

TIME: 0:02.07 CPU: 86% major: 0 minor: 8207 invol: 28 vol: 129
TIME: 0:01.73 CPU: 99% major: 0 minor: 8208 invol: 10 vol: 5

TIME: 0:01.82 CPU: 100% major: 0 minor: 8206 invol: 16 vol: 4
TIME: 0:01.69 CPU: 100% major: 0 minor: 8207 invol: 8 vol: 4

TIME: 0:01.76 CPU: 99% major: 0 minor: 8207 invol: 10 vol: 4
TIME: 0:01.76 CPU: 100% major: 0 minor: 8208 invol: 6 vol: 4

TIME: 0:02.00 CPU: 99% major: 0 minor: 8208 invol: 7 vol: 4
TIME: 0:01.86 CPU: 100% major: 0 minor: 8207 invol: 10 vol: 5

TIME: 0:01.74 CPU: 100% major: 0 minor: 8207 invol: 9 vol: 4
TIME: 0:01.69 CPU: 100% major: 0 minor: 8208 invol: 11 vol: 4

1000000

/dev/md2 on /raid/heinrich type reiserfs (rw,noexec,nosuid,nodev)
100 (1*4 KB) Dateien on reiserfs

---

time create_testfiles2.sh V_100:
TIME: 0:00.37 CPU: 53% major: 7 minor: 22307 invol: 181 vol: 206

time ctmk c_100.ct:
TIME: 0:00.02 CPU: 50% major: 0 minor: 135 invol: 0 vol: 110

---

avtest V_100:

TIME: 0:00.03 CPU: 31% major: 0 minor: 132 invol: 1 vol: 109
TIME: 0:00.00 CPU: 100% major: 0 minor: 131 invol: 1 vol: 2

TIME: 0:00.04 CPU: 19% major: 0 minor: 131 invol: 1 vol: 105
TIME: 0:00.00 CPU: 100% major: 0 minor: 131 invol: 1 vol: 2

TIME: 0:00.02 CPU: 64% major: 0 minor: 131 invol: 0 vol: 105
TIME: 0:00.00 CPU: 100% major: 0 minor: 132 invol: 1 vol: 2

TIME: 0:00.03 CPU: 36% major: 0 minor: 130 invol: 1 vol: 105
TIME: 0:00.00 CPU: 100% major: 0 minor: 132 invol: 1 vol: 2

TIME: 0:00.03 CPU: 36% major: 0 minor: 130 invol: 0 vol: 105
TIME: 0:00.00 CPU: 100% major: 0 minor: 132 invol: 1 vol: 2

100

---

actest c_100.ct:

TIME: 0:00.01 CPU: 28% major: 1 minor: 135 invol: 2 vol: 12
TIME: 0:00.00 CPU: 133% major: 0 minor: 135 invol: 1 vol: 2

TIME: 0:00.01 CPU: 0% major: 0 minor: 136 invol: 0 vol: 8
TIME: 0:00.00 CPU: 0% major: 0 minor: 135 invol: 1 vol: 2

TIME: 0:00.01 CPU: 40% major: 0 minor: 135 invol: 1 vol: 8
TIME: 0:00.00 CPU: 0% major: 0 minor: 135 invol: 1 vol: 2

TIME: 0:00.01 CPU: 92% major: 0 minor: 136 invol: 1 vol: 8
TIME: 0:00.00 CPU: 0% major: 0 minor: 136 invol: 1 vol: 2

```
TIME: 0:00.01  CPU: 0%    major: 0  minor: 135    invol: 0   vol: 10
TIME: 0:00.00  CPU: 133%  major: 0  minor: 136    invol: 0   vol: 2

100
-------------------------------------------------------------------------------

/dev/md2 on /raid/heinrich type ext3 (rw,noexec,nosuid,nodev)
1000 (1*4 KB) Dateien on ext3
-------------------------------------------------------------------------------

time create_testfiles2.sh V_1000:
TIME: 0:01.95  CPU: 97%   major: 0  minor: 214586   invol: 1720  vol: 1737

time ctmk c_1000.ct:
TIME: 0:00.22  CPU: 63%   major: 0  minor: 184      invol: 6   vol: 1048

-------------------------------------------------------------------------------

avtest V_1000:

TIME: 0:00.90  CPU: 6%    major: 0  minor: 143      invol: 4   vol: 1045
TIME: 0:00.03  CPU: 102%  major: 0  minor: 144      invol: 1   vol: 2

TIME: 0:00.67  CPU: 12%   major: 0  minor: 142      invol: 0   vol: 1044
TIME: 0:00.03  CPU: 105%  major: 0  minor: 143      invol: 1   vol: 2

TIME: 0:00.58  CPU: 12%   major: 0  minor: 143      invol: 1   vol: 1044
TIME: 0:00.03  CPU: 93%   major: 0  minor: 143      invol: 0   vol: 2

TIME: 0:00.61  CPU: 10%   major: 0  minor: 145      invol: 1   vol: 1044
TIME: 0:00.03  CPU: 93%   major: 0  minor: 144      invol: 1   vol: 2

TIME: 0:00.74  CPU: 8%    major: 0  minor: 143      invol: 2   vol: 1044
TIME: 0:00.03  CPU: 100%  major: 0  minor: 143      invol: 1   vol: 2

1000
-------------------------------------------------------------------------------

actest c_1000.ct:

TIME: 0:00.12  CPU: 9%    major: 0  minor: 149      invol: 1   vol: 22
TIME: 0:00.01  CPU: 100%  major: 0  minor: 150      invol: 0   vol: 2

TIME: 0:00.05  CPU: 22%   major: 0  minor: 149      invol: 1   vol: 33
TIME: 0:00.01  CPU: 100%  major: 0  minor: 149      invol: 0   vol: 2

TIME: 0:00.05  CPU: 22%   major: 0  minor: 148      invol: 1   vol: 25
TIME: 0:00.01  CPU: 100%  major: 0  minor: 149      invol: 1   vol: 2

TIME: 0:00.05  CPU: 28%   major: 0  minor: 149      invol: 2   vol: 24
TIME: 0:00.01  CPU: 100%  major: 0  minor: 150      invol: 1   vol: 2

TIME: 0:00.06  CPU: 32%   major: 0  minor: 149      invol: 1   vol: 25
TIME: 0:00.01  CPU: 100%  major: 0  minor: 149      invol: 1   vol: 2

1000
-------------------------------------------------------------------------------

/dev/md2 on /raid/heinrich type ext3 (rw,noexec,nosuid,nodev)
10000 (1*4 KB) Dateien on ext3
-------------------------------------------------------------------------------

time create_testfiles2.sh V_10000:
TIME: 0:19.19  CPU: 97%   major: 0  minor: 2139429  invol: 17322  vol: 17361

time ctmk c_10000.ct:
TIME: 0:03.06  CPU: 39%   major: 0  minor: 498      invol: 8   vol: 10399

-------------------------------------------------------------------------------

avtest V_10000:

TIME: 0:04.31  CPU: 15%   major: 0  minor: 286      invol: 7   vol: 10397
TIME: 0:00.32  CPU: 95%   major: 0  minor: 286      invol: 11  vol: 2

TIME: 0:04.26  CPU: 16%   major: 0  minor: 286      invol: 9   vol: 10397
TIME: 0:00.32  CPU: 95%   major: 0  minor: 286      invol: 12  vol: 7

TIME: 0:02.94  CPU: 23%   major: 0  minor: 285      invol: 7   vol: 10394
TIME: 0:00.33  CPU: 96%   major: 0  minor: 286      invol: 11  vol: 2

TIME: 0:07.17  CPU: 10%   major: 0  minor: 286      invol: 52  vol: 10395
TIME: 0:00.32  CPU: 95%   major: 0  minor: 285      invol: 12  vol: 2

TIME: 0:02.45  CPU: 31%   major: 0  minor: 286      invol: 8   vol: 10394
TIME: 0:00.32  CPU: 96%   major: 0  minor: 286      invol: 12  vol: 2

10000
-------------------------------------------------------------------------------

actest c_10000.ct:

TIME: 0:00.55  CPU: 32%   major: 0  minor: 295      invol: 4   vol: 133
TIME: 0:00.16  CPU: 98%   major: 0  minor: 295      invol: 1   vol: 2

TIME: 0:00.42  CPU: 43%   major: 0  minor: 295      invol: 10  vol: 160
TIME: 0:00.15  CPU: 98%   major: 0  minor: 294      invol: 6   vol: 2

TIME: 0:00.39  CPU: 40%   major: 0  minor: 294      invol: 11  vol: 161
TIME: 0:00.15  CPU: 99%   major: 0  minor: 294      invol: 2   vol: 2

TIME: 0:00.40  CPU: 34%   major: 0  minor: 294      invol: 10  vol: 151
TIME: 0:00.15  CPU: 98%   major: 0  minor: 293      invol: 1   vol: 2

TIME: 0:00.38  CPU: 42%   major: 0  minor: 294      invol: 9   vol: 166
TIME: 0:00.15  CPU: 99%   major: 0  minor: 295      invol: 3   vol: 2

10000
```

```
TIME: 0:00.01  CPU: 36%   major: 0  minor: 135    invol: 0   vol: 8
TIME: 0:00.00  CPU: 133%  major: 0  minor: 135    invol: 1   vol: 2

100
-------------------------------------------------------------------------------

/dev/md2 on /raid/heinrich type reiserfs (rw,noexec,nosuid,nodev)
1000 (1*4 KB) Dateien on reiserfs
-------------------------------------------------------------------------------

time create_testfiles2.sh V_1000:
TIME: 0:02.35  CPU: 75%   major: 0  minor: 215616   invol: 1792  vol: 1789

time ctmk c_1000.ct:
TIME: 0:00.23  CPU: 54%   major: 0  minor: 184      invol: 0   vol: 1029

-------------------------------------------------------------------------------

avtest V_1000:

TIME: 0:00.20  CPU: 30%   major: 0  minor: 144      invol: 1   vol: 1036
TIME: 0:00.02  CPU: 96%   major: 0  minor: 144      invol: 1   vol: 2

TIME: 0:00.18  CPU: 40%   major: 0  minor: 143      invol: 1   vol: 1035
TIME: 0:00.02  CPU: 96%   major: 0  minor: 143      invol: 1   vol: 2

TIME: 0:00.19  CPU: 32%   major: 0  minor: 143      invol: 22  vol: 1036
TIME: 0:00.03  CPU: 90%   major: 0  minor: 143      invol: 0   vol: 2

TIME: 0:00.19  CPU: 32%   major: 0  minor: 143      invol: 0   vol: 1035
TIME: 0:00.02  CPU: 96%   major: 0  minor: 142      invol: 1   vol: 2

TIME: 0:00.18  CPU: 46%   major: 0  minor: 143      invol: 2   vol: 1035
TIME: 0:00.03  CPU: 90%   major: 0  minor: 144      invol: 1   vol: 2

1000
-------------------------------------------------------------------------------

actest c_1000.ct:

TIME: 0:00.04  CPU: 42%   major: 0  minor: 148      invol: 5   vol: 28
TIME: 0:00.01  CPU: 100%  major: 0  minor: 150      invol: 0   vol: 2

TIME: 0:00.05  CPU: 76%   major: 0  minor: 149      invol: 3   vol: 24
TIME: 0:00.01  CPU: 100%  major: 0  minor: 150      invol: 0   vol: 2

TIME: 0:00.04  CPU: 40%   major: 0  minor: 149      invol: 3   vol: 32
TIME: 0:00.01  CPU: 100%  major: 0  minor: 150      invol: 0   vol: 2

TIME: 0:00.05  CPU: 37%   major: 0  minor: 149      invol: 2   vol: 35
TIME: 0:00.01  CPU: 100%  major: 0  minor: 150      invol: 0   vol: 2

TIME: 0:00.05  CPU: 50%   major: 0  minor: 149      invol: 2   vol: 31
TIME: 0:00.01  CPU: 100%  major: 0  minor: 149      invol: 0   vol: 2

1000
-------------------------------------------------------------------------------

/dev/md2 on /raid/heinrich type reiserfs (rw,noexec,nosuid,nodev)
10000 (1*4 KB) Dateien on reiserfs
-------------------------------------------------------------------------------

time create_testfiles2.sh V_10000:
TIME: 0:19.89  CPU: 96%   major: 0  minor: 2139437  invol: 17737  vol: 17662

time ctmk c_10000.ct:
TIME: 0:02.32  CPU: 57%   major: 0  minor: 470      invol: 10  vol: 10251

-------------------------------------------------------------------------------

avtest V_10000:

TIME: 0:01.79  CPU: 45%   major: 0  minor: 286      invol: 9   vol: 10323
TIME: 0:00.35  CPU: 96%   major: 0  minor: 285      invol: 13  vol: 2

TIME: 0:01.76  CPU: 40%   major: 0  minor: 286      invol: 25  vol: 10320
TIME: 0:00.34  CPU: 95%   major: 0  minor: 286      invol: 12  vol: 2

TIME: 0:01.78  CPU: 43%   major: 0  minor: 285      invol: 10  vol: 10320
TIME: 0:00.37  CPU: 97%   major: 0  minor: 287      invol: 9   vol: 2

TIME: 0:01.78  CPU: 43%   major: 0  minor: 286      invol: 28  vol: 10323
TIME: 0:00.35  CPU: 96%   major: 0  minor: 287      invol: 12  vol: 2

TIME: 0:01.75  CPU: 44%   major: 0  minor: 286      invol: 13  vol: 10320
TIME: 0:00.34  CPU: 97%   major: 0  minor: 287      invol: 8   vol: 2

10000
-------------------------------------------------------------------------------

actest c_10000.ct:

TIME: 0:00.46  CPU: 37%   major: 0  minor: 296      invol: 14  vol: 139
TIME: 0:00.14  CPU: 99%   major: 0  minor: 294      invol: 2   vol: 5

TIME: 0:00.40  CPU: 47%   major: 0  minor: 294      invol: 15  vol: 157
TIME: 0:00.14  CPU: 100%  major: 0  minor: 295      invol: 2   vol: 2

TIME: 0:00.41  CPU: 51%   major: 0  minor: 294      invol: 14  vol: 168
TIME: 0:00.14  CPU: 100%  major: 0  minor: 294      invol: 1   vol: 6

TIME: 0:00.45  CPU: 47%   major: 0  minor: 295      invol: 13  vol: 162
TIME: 0:00.14  CPU: 100%  major: 0  minor: 294      invol: 1   vol: 2

TIME: 0:00.41  CPU: 53%   major: 0  minor: 294      invol: 13  vol: 168
TIME: 0:00.14  CPU: 100%  major: 0  minor: 295      invol: 2   vol: 2

10000
```

/dev/md2 on /raid/heinrich type ext3 (rw,noexec,nosuid,nodev)
100000 (1*4 KB) Dateien on ext3

---

time create_testfiles2.sh V_100000:
TIME: 3:15.53 CPU: 92% major: 0 minor: 21488304 invol: 175186 vol: 174669

time ctmk c_100000.ct:
TIME: 0:45.43 CPU: 27% major: 0 minor: 3698 invol: 2206 vol: 104026

---

avtest V_100000:

TIME: 0:46.38 CPU: 15% major: 0 minor: 1691 invol: 724 vol: 104015
TIME: 0:03.36 CPU: 95% major: 0 minor: 1692 invol: 103 vol: 6

TIME: 0:33.56 CPU: 21% major: 0 minor: 1692 invol: 426 vol: 104010
TIME: 0:03.45 CPU: 95% major: 0 minor: 1693 invol: 100 vol: 6

TIME: 0:36.47 CPU: 21% major: 0 minor: 1693 invol: 372 vol: 104008
TIME: 0:03.41 CPU: 96% major: 0 minor: 1692 invol: 98 vol: 6

TIME: 0:40.92 CPU: 17% major: 0 minor: 1692 invol: 558 vol: 104010
TIME: 0:03.47 CPU: 98% major: 0 minor: 1693 invol: 62 vol: 7

TIME: 0:31.72 CPU: 23% major: 0 minor: 1694 invol: 549 vol: 104010
TIME: 0:03.42 CPU: 99% major: 0 minor: 1692 invol: 54 vol: 6

100000

---

actest c_100000.ct:

TIME: 0:03.82 CPU: 39% major: 0 minor: 1722 invol: 112 vol: 1526
TIME: 0:01.60 CPU: 99% major: 0 minor: 1723 invol: 13 vol: 2

TIME: 0:03.75 CPU: 40% major: 0 minor: 1722 invol: 118 vol: 1600
TIME: 0:01.60 CPU: 100% major: 0 minor: 1723 invol: 10 vol: 2

TIME: 0:03.79 CPU: 37% major: 0 minor: 1722 invol: 113 vol: 1566
TIME: 0:01.59 CPU: 99% major: 0 minor: 1723 invol: 15 vol: 2

TIME: 0:03.82 CPU: 38% major: 0 minor: 1721 invol: 114 vol: 1491
TIME: 0:01.60 CPU: 100% major: 0 minor: 1723 invol: 9 vol: 2

TIME: 0:03.79 CPU: 38% major: 0 minor: 1723 invol: 131 vol: 1574
TIME: 0:01.65 CPU: 100% major: 0 minor: 1722 invol: 14 vol: 2

100000

/dev/md2 on /raid/heinrich type ext3 (rw,noexec,nosuid,nodev)
1000000 (1*4 KB) Dateien on ext3

---

time create_testfiles2.sh V_1000000:
TIME: 35:21.43 CPU: 77% major: 0 minor: 214873236 invol: 1785838
vol: 1784881

time ctmk c_1000000.ct:
TIME: 10:29.30 CPU: 25% major: 0 minor: 35571 invol: 23535 vol: 1070895

---

avtest V_1000000:

TIME: 10:16.19 CPU: 12% major: 0 minor: 15754 invol: 10817 vol: 1039920
TIME: 5:53.03 CPU: 22% major: 0 minor: 15755 invol: 7871 vol: 1031310

TIME: 8:48.68 CPU: 15% major: 0 minor: 15755 invol: 8202 vol: 1039894
TIME: 7:04.67 CPU: 18% major: 0 minor: 15754 invol: 9742 vol: 1031306

TIME: 8:53.07 CPU: 15% major: 0 minor: 15755 invol: 9314 vol: 1039896
TIME: 5:53.73 CPU: 22% major: 0 minor: 15755 invol: 8105 vol: 1031299

TIME: 7:59.79 CPU: 16% major: 0 minor: 15755 invol: 8606 vol: 1039876
TIME: 5:50.52 CPU: 22% major: 0 minor: 15756 invol: 8681 vol: 1031303

TIME: 8:10.08 CPU: 16% major: 0 minor: 15756 invol: 8646 vol: 1039881
TIME: 7:19.51 CPU: 18% major: 0 minor: 15755 invol: 9512 vol: 1031316

1000000

---

actest c_1000000.ct:

TIME: 0:38.47 CPU: 45% major: 0 minor: 16004 invol: 2459 vol: 15249
TIME: 0:38.75 CPU: 40% major: 1 minor: 16002 invol: 2774 vol: 15358

TIME: 0:38.56 CPU: 41% major: 1 minor: 16003 invol: 2380 vol: 15107
TIME: 0:38.86 CPU: 39% major: 1 minor: 16003 invol: 2768 vol: 15208

TIME: 0:38.78 CPU: 39% major: 1 minor: 16004 invol: 2431 vol: 14952
TIME: 0:38.49 CPU: 41% major: 1 minor: 16004 invol: 2755 vol: 15237

TIME: 0:38.70 CPU: 39% major: 1 minor: 16003 invol: 2415 vol: 15057
TIME: 0:38.16 CPU: 45% major: 1 minor: 16002 invol: 2801 vol: 15443

TIME: 0:38.69 CPU: 40% major: 1 minor: 16004 invol: 2472 vol: 15010
TIME: 0:38.78 CPU: 40% major: 1 minor: 16003 invol: 2784 vol: 15124

1000000

/dev/md2 on /raid/heinrich type ext3 (rw,noexec,nosuid,nodev)
10 (256*4 KB) Dateien on ext3

/dev/md2 on /raid/heinrich type reiserfs (rw,noexec,nosuid,nodev)
100000 (1*4 KB) Dateien on reiserfs

---

time create_testfiles2.sh V_100000:
TIME: 3:21.11 CPU: 98% major: 0 minor: 21487779 invol: 178881 vol: 177763

time ctmk c_100000.ct:
TIME: 0:27.67 CPU: 49% major: 0 minor: 3451 invol: 589 vol: 102484

---

avtest V_100000:

TIME: 0:18.66 CPU: 43% major: 0 minor: 1691 invol: 185 vol: 103209
TIME: 0:03.77 CPU: 95% major: 0 minor: 1692 invol: 118 vol: 26

TIME: 0:18.04 CPU: 42% major: 0 minor: 1694 invol: 88 vol: 103209
TIME: 0:03.82 CPU: 95% major: 0 minor: 1692 invol: 120 vol: 22

TIME: 0:18.38 CPU: 44% major: 0 minor: 1692 invol: 169 vol: 103209
TIME: 0:03.75 CPU: 95% major: 0 minor: 1692 invol: 134 vol: 20

TIME: 0:18.60 CPU: 43% major: 0 minor: 1692 invol: 175 vol: 103208
TIME: 0:03.77 CPU: 95% major: 0 minor: 1691 invol: 107 vol: 23

TIME: 0:19.07 CPU: 41% major: 0 minor: 1692 invol: 234 vol: 103206
TIME: 0:03.80 CPU: 95% major: 0 minor: 1692 invol: 112 vol: 23

100000

---

actest c_100000.ct:

TIME: 0:04.11 CPU: 44% major: 0 minor: 1723 invol: 208 vol: 1469
TIME: 0:01.65 CPU: 100% major: 0 minor: 1721 invol: 14 vol: 7

TIME: 0:03.97 CPU: 47% major: 0 minor: 1722 invol: 234 vol: 1475
TIME: 0:01.51 CPU: 99% major: 0 minor: 1721 invol: 9 vol: 2

TIME: 0:03.92 CPU: 47% major: 0 minor: 1722 invol: 217 vol: 1475
TIME: 0:01.51 CPU: 99% major: 0 minor: 1722 invol: 8 vol: 2

TIME: 0:03.99 CPU: 46% major: 0 minor: 1723 invol: 218 vol: 1455
TIME: 0:01.51 CPU: 99% major: 0 minor: 1722 invol: 5 vol: 2

TIME: 0:03.97 CPU: 45% major: 0 minor: 1724 invol: 213 vol: 1425
TIME: 0:01.53 CPU: 100% major: 0 minor: 1722 invol: 3 vol: 6

100000

/dev/md2 on /raid/heinrich type reiserfs (rw,noexec,nosuid,nodev)
1000000 (1*4 KB) Dateien on reiserfs

---

time create_testfiles2.sh V_1000000:
TIME: 33:40.29 CPU: 109% major: 0 minor: 214873354 invol: 1872962
vol: 1841901

time ctmk c_1000000.ct:
TIME: 6:02.60 CPU: 48% major: 0 minor: 33114 invol: 11881 vol: 1054894

---

avtest V_1000000:

TIME: 3:31.25 CPU: 40% major: 0 minor: 15756 invol: 4566 vol: 1032021
TIME: 3:27.75 CPU: 41% major: 0 minor: 15754 invol: 4296 vol: 1032046

TIME: 3:27.13 CPU: 40% major: 0 minor: 15756 invol: 3828 vol: 1032019
TIME: 3:32.64 CPU: 40% major: 0 minor: 15756 invol: 4803 vol: 1032043

TIME: 3:31.48 CPU: 40% major: 0 minor: 15755 invol: 3977 vol: 1032033
TIME: 3:29.34 CPU: 40% major: 0 minor: 15755 invol: 4459 vol: 1032081

TIME: 3:29.31 CPU: 40% major: 0 minor: 15755 invol: 4096 vol: 1032038
TIME: 3:30.01 CPU: 40% major: 0 minor: 15755 invol: 5092 vol: 1032004

TIME: 3:29.95 CPU: 39% major: 0 minor: 15756 invol: 4076 vol: 1032023
TIME: 3:30.44 CPU: 41% major: 0 minor: 15755 invol: 4846 vol: 1032065

1000000

---

actest c_1000000.ct:

TIME: 0:39.97 CPU: 52% major: 0 minor: 16004 invol: 3511 vol: 14300
TIME: 0:40.67 CPU: 53% major: 1 minor: 16003 invol: 3896 vol: 13672

TIME: 0:39.87 CPU: 52% major: 1 minor: 16004 invol: 3583 vol: 14252
TIME: 0:39.64 CPU: 52% major: 1 minor: 16004 invol: 3914 vol: 14393

TIME: 0:39.76 CPU: 51% major: 1 minor: 16003 invol: 3508 vol: 14112
TIME: 0:39.81 CPU: 53% major: 1 minor: 16003 invol: 3942 vol: 14312

TIME: 0:39.68 CPU: 52% major: 1 minor: 16003 invol: 3611 vol: 14255
TIME: 0:39.71 CPU: 52% major: 1 minor: 16005 invol: 3957 vol: 14438

TIME: 0:39.71 CPU: 51% major: 1 minor: 16004 invol: 3535 vol: 14185
TIME: 0:39.77 CPU: 52% major: 1 minor: 16003 invol: 3956 vol: 14310

1000000

/dev/md2 on /raid/heinrich type reiserfs (rw,noexec,nosuid,nodev)
10 (256*4 KB) Dateien on reiserfs

---

time create_testfiles2.sh V_10:
TIME: 0:00.59 CPU: 76% major: 7 minor: 2866 invol: 28 vol: 49

time ctmk c_10.ct:
TIME: 0:00.32 CPU: 31% major: 0 minor: 132 invol: 4 vol: 73

time tar -cvf V_10:
tar: Removing leading `/' from member names
TIME: 0:00.36 CPU: 22% major: 4 minor: 214 invol: 9 vol: 74

---

time ls -l V_10:

TIME: 0:00.01 CPU: 0% major: 1 minor: 218 invol: 4 vol: 9
TIME: 0:00.00 CPU: 200% major: 0 minor: 219 invol: 3 vol: 4

TIME: 0:00.00 CPU: 0% major: 0 minor: 219 invol: 4 vol: 4
TIME: 0:00.00 CPU: 0% major: 0 minor: 218 invol: 2 vol: 5

TIME: 0:00.00 CPU: 200% major: 0 minor: 221 invol: 3 vol: 6
TIME: 0:00.00 CPU: 0% major: 0 minor: 221 invol: 5 vol: 1

TIME: 0:00.00 CPU: 200% major: 0 minor: 220 invol: 1 vol: 8
TIME: 0:00.00 CPU: 0% major: 0 minor: 220 invol: 4 vol: 3

TIME: 0:00.00 CPU: 200% major: 0 minor: 219 invol: 0 vol: 8
TIME: 0:00.00 CPU: 200% major: 0 minor: 221 invol: 5 vol: 1

10

---

time tar -tf T_10.tar:

TIME: 0:00.10 CPU: 7% major: 0 minor: 197 invol: 1 vol: 31
TIME: 0:00.01 CPU: 109% major: 0 minor: 198 invol: 1 vol: 1

TIME: 0:00.12 CPU: 6% major: 0 minor: 198 invol: 0 vol: 31
TIME: 0:00.01 CPU: 109% major: 0 minor: 197 invol: 1 vol: 1

TIME: 0:00.12 CPU: 12% major: 0 minor: 196 invol: 1 vol: 32
TIME: 0:00.01 CPU: 109% major: 0 minor: 196 invol: 1 vol: 1

TIME: 0:00.11 CPU: 20% major: 0 minor: 198 invol: 2 vol: 31
TIME: 0:00.01 CPU: 109% major: 0 minor: 198 invol: 1 vol: 1

TIME: 0:00.12 CPU: 9% major: 0 minor: 198 invol: 1 vol: 36
TIME: 0:00.01 CPU: 109% major: 0 minor: 198 invol: 1 vol: 1

11

---

time ctls c_10.ct:

TIME: 0:00.02 CPU: 0% major: 0 minor: 122 invol: 1 vol: 8
TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 1 vol: 2

TIME: 0:00.02 CPU: 0% major: 0 minor: 122 invol: 0 vol: 7
TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 1 vol: 2

TIME: 0:00.01 CPU: 0% major: 0 minor: 121 invol: 0 vol: 7
TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 1 vol: 2

TIME: 0:00.00 CPU: 200% major: 0 minor: 120 invol: 1 vol: 7
TIME: 0:00.00 CPU: 0% major: 0 minor: 122 invol: 1 vol: 2

TIME: 0:00.00 CPU: 0% major: 0 minor: 122 invol: 0 vol: 7
TIME: 0:00.00 CPU: 0% major: 0 minor: 122 invol: 1 vol: 2

10

---

time ls -l V_10:

TIME: 0:00.02 CPU: 0% major: 0 minor: 239 invol: 4 vol: 6
TIME: 0:00.00 CPU: 200% major: 0 minor: 238 invol: 3 vol: 3

TIME: 0:00.01 CPU: 25% major: 0 minor: 237 invol: 3 vol: 6
TIME: 0:00.00 CPU: 200% major: 0 minor: 238 invol: 3 vol: 3

TIME: 0:00.02 CPU: 14% major: 0 minor: 238 invol: 3 vol: 6
TIME: 0:00.00 CPU: 200% major: 0 minor: 238 invol: 4 vol: 3

TIME: 0:00.04 CPU: 0% major: 0 minor: 238 invol: 3 vol: 6
TIME: 0:00.00 CPU: 200% major: 0 minor: 237 invol: 1 vol: 5

TIME: 0:00.00 CPU: 133% major: 0 minor: 240 invol: 5 vol: 4
TIME: 0:00.00 CPU: 200% major: 0 minor: 238 invol: 5 vol: 1

11

---

time tar -tvf T_10.tar:

TIME: 0:00.13 CPU: 8% major: 0 minor: 208 invol: 1 vol: 34
TIME: 0:00.01 CPU: 109% major: 0 minor: 209 invol: 1 vol: 1

TIME: 0:00.16 CPU: 14% major: 0 minor: 208 invol: 1 vol: 38
TIME: 0:00.01 CPU: 109% major: 0 minor: 208 invol: 1 vol: 1

TIME: 0:00.11 CPU: 31% major: 0 minor: 209 invol: 3 vol: 35
TIME: 0:00.01 CPU: 109% major: 0 minor: 209 invol: 1 vol: 1

TIME: 0:00.11 CPU: 10% major: 0 minor: 208 invol: 0 vol: 32
TIME: 0:00.01 CPU: 109% major: 0 minor: 210 invol: 1 vol: 1

TIME: 0:00.11 CPU: 21% major: 0 minor: 209 invol: 1 vol: 31
TIME: 0:00.01 CPU: 109% major: 0 minor: 208 invol: 1 vol: 1

---

time create_testfiles2.sh V_10:
TIME: 0:00.77 CPU: 60% major: 7 minor: 2865 invol: 27 vol: 49

time ctmk c_10.ct:
TIME: 0:00.16 CPU: 61% major: 0 minor: 132 invol: 12 vol: 61

time tar -cvf V_10:
tar: Removing leading `/' from member names
TIME: 0:00.20 CPU: 27% major: 4 minor: 213 invol: 3 vol: 59

---

time ls -l V_10:

TIME: 0:00.01 CPU: 40% major: 1 minor: 217 invol: 5 vol: 4
TIME: 0:00.00 CPU: 200% major: 0 minor: 220 invol: 4 vol: 2

TIME: 0:00.00 CPU: 0% major: 0 minor: 220 invol: 1 vol: 5
TIME: 0:00.00 CPU: 0% major: 0 minor: 220 invol: 1 vol: 5

TIME: 0:00.00 CPU: 0% major: 0 minor: 219 invol: 3 vol: 2
TIME: 0:00.00 CPU: 0% major: 0 minor: 221 invol: 5 vol: 1

TIME: 0:00.00 CPU: 0% major: 0 minor: 218 invol: 4 vol: 1
TIME: 0:00.00 CPU: 0% major: 0 minor: 220 invol: 5 vol: 1

TIME: 0:00.00 CPU: 0% major: 0 minor: 219 invol: 4 vol: 1
TIME: 0:00.00 CPU: 200% major: 0 minor: 218 invol: 4 vol: 3

10

---

time tar -tf T_10.tar:

TIME: 0:00.10 CPU: 26% major: 0 minor: 197 invol: 1 vol: 27
TIME: 0:00.01 CPU: 109% major: 0 minor: 197 invol: 1 vol: 1

TIME: 0:00.11 CPU: 31% major: 0 minor: 198 invol: 1 vol: 38
TIME: 0:00.01 CPU: 72% major: 0 minor: 197 invol: 1 vol: 1

TIME: 0:00.12 CPU: 10% major: 0 minor: 197 invol: 2 vol: 28
TIME: 0:00.01 CPU: 109% major: 0 minor: 197 invol: 1 vol: 1

TIME: 0:00.12 CPU: 28% major: 0 minor: 196 invol: 2 vol: 46
TIME: 0:00.01 CPU: 72% major: 0 minor: 199 invol: 1 vol: 1

TIME: 0:00.11 CPU: 28% major: 0 minor: 198 invol: 3 vol: 47
TIME: 0:00.01 CPU: 72% major: 0 minor: 198 invol: 1 vol: 1

11

---

time ctls c_10.ct:

TIME: 0:00.00 CPU: 0% major: 0 minor: 122 invol: 1 vol: 6
TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 1 vol: 3

TIME: 0:00.01 CPU: 36% major: 0 minor: 121 invol: 1 vol: 6
TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 1 vol: 2

TIME: 0:00.00 CPU: 0% major: 0 minor: 123 invol: 0 vol: 6
TIME: 0:00.00 CPU: 0% major: 0 minor: 121 invol: 0 vol: 2

TIME: 0:00.00 CPU: 0% major: 0 minor: 122 invol: 1 vol: 6
TIME: 0:00.00 CPU: 400% major: 0 minor: 121 invol: 1 vol: 2

TIME: 0:00.00 CPU: 50% major: 0 minor: 120 invol: 1 vol: 6
TIME: 0:00.00 CPU: 400% major: 0 minor: 122 invol: 1 vol: 2

10

---

time ls -l V_10:

TIME: 0:00.00 CPU: 133% major: 0 minor: 238 invol: 1 vol: 7
TIME: 0:00.00 CPU: 0% major: 0 minor: 238 invol: 4 vol: 3

TIME: 0:00.00 CPU: 133% major: 0 minor: 239 invol: 1 vol: 7
TIME: 0:00.00 CPU: 200% major: 0 minor: 238 invol: 1 vol: 5

TIME: 0:00.00 CPU: 0% major: 0 minor: 238 invol: 3 vol: 5
TIME: 0:00.00 CPU: 0% major: 0 minor: 238 invol: 1 vol: 5

TIME: 0:00.00 CPU: 133% major: 0 minor: 240 invol: 4 vol: 3
TIME: 0:00.00 CPU: 0% major: 0 minor: 238 invol: 5 vol: 1

TIME: 0:00.00 CPU: 133% major: 0 minor: 238 invol: 4 vol: 3
TIME: 0:00.00 CPU: 0% major: 0 minor: 238 invol: 4 vol: 3

11

---

time tar -tvf T_10.tar:

TIME: 0:00.11 CPU: 20% major: 0 minor: 207 invol: 6 vol: 29
TIME: 0:00.01 CPU: 72% major: 0 minor: 209 invol: 1 vol: 1

TIME: 0:00.13 CPU: 11% major: 0 minor: 208 invol: 2 vol: 29
TIME: 0:00.01 CPU: 72% major: 0 minor: 207 invol: 1 vol: 1

TIME: 0:00.12 CPU: 20% major: 0 minor: 207 invol: 3 vol: 31
TIME: 0:00.01 CPU: 72% major: 0 minor: 208 invol: 1 vol: 1

TIME: 0:00.12 CPU: 21% major: 0 minor: 209 invol: 4 vol: 29
TIME: 0:00.01 CPU: 109% major: 0 minor: 208 invol: 1 vol: 1

TIME: 0:00.12 CPU: 22% major: 0 minor: 210 invol: 2 vol: 29
TIME: 0:00.01 CPU: 109% major: 0 minor: 209 invol: 1 vol: 1

```
11
-------------------------------------------------------------------------------

time ctls -l c_10.ct:

TIME: 0:00.02  CPU: 0%    major: 0    minor: 149    invol: 5    vol: 7
TIME: 0:00.00  CPU: 0%    major: 0    minor: 149    invol: 3    vol: 6

TIME: 0:00.00  CPU: 0%    major: 0    minor: 149    invol: 3    vol: 4
TIME: 0:00.00  CPU: 0%    major: 0    minor: 149    invol: 4    vol: 4

TIME: 0:00.00  CPU: 0%    major: 0    minor: 149    invol: 4    vol: 4
TIME: 0:00.00  CPU: 0%    major: 0    minor: 149    invol: 3    vol: 4

TIME: 0:00.00  CPU: 0%    major: 0    minor: 149    invol: 4    vol: 4
TIME: 0:00.00  CPU: 0%    major: 0    minor: 149    invol: 4    vol: 4

TIME: 0:00.00  CPU: 0%    major: 0    minor: 148    invol: 3    vol: 4
TIME: 0:00.00  CPU: 0%    major: 0    minor: 149    invol: 4    vol: 4

10
-------------------------------------------------------------------------------

/dev/md2 on /raid/heinrich type ext3 (rw,noexec,nosuid,nodev)
100 (256*4 KB) Dateien on ext3
-------------------------------------------------------------------------------

time create_testfiles2.sh V_100:
TIME: 0:01.32  CPU: 98%   major: 0    minor: 22099    invol: 214    vol: 212

time ctmk c_100.ct:
TIME: 0:03.44  CPU: 28%   major: 0    minor: 136    invol: 62    vol: 577

time tar -cvf V_100:
tar: Removing leading `/' from member names
TIME: 0:02.98  CPU: 24%   major: 0    minor: 219    invol: 36    vol: 592
-------------------------------------------------------------------------------

time ls -l V_100:

TIME: 0:00.01  CPU: 22%   major: 0    minor: 224    invol: 5    vol: 4
TIME: 0:00.00  CPU: 400%  major: 0    minor: 225    invol: 4    vol: 3

TIME: 0:00.00  CPU: 133%  major: 0    minor: 225    invol: 2    vol: 6
TIME: 0:00.00  CPU: 400%  major: 0    minor: 225    invol: 4    vol: 2

TIME: 0:00.00  CPU: 0%    major: 0    minor: 224    invol: 5    vol: 4
TIME: 0:00.00  CPU: 200%  major: 0    minor: 223    invol: 3    vol: 5

TIME: 0:00.00  CPU: 0%    major: 0    minor: 224    invol: 0    vol: 8
TIME: 0:00.00  CPU: 200%  major: 0    minor: 224    invol: 5    vol: 1

TIME: 0:00.00  CPU: 133%  major: 0    minor: 225    invol: 2    vol: 6
TIME: 0:00.00  CPU: 200%  major: 0    minor: 224    invol: 3    vol: 3

100
-------------------------------------------------------------------------------

time tar -tf T_100.tar:

TIME: 0:01.02  CPU: 14%   major: 0    minor: 197    invol: 5    vol: 318
TIME: 0:00.09  CPU: 105%  major: 0    minor: 198    invol: 2    vol: 1

TIME: 0:01.05  CPU: 13%   major: 0    minor: 198    invol: 5    vol: 264
TIME: 0:00.09  CPU: 100%  major: 0    minor: 198    invol: 1    vol: 4

TIME: 0:01.05  CPU: 13%   major: 0    minor: 199    invol: 9    vol: 259
TIME: 0:00.10  CPU: 99%   major: 0    minor: 198    invol: 2    vol: 1

TIME: 0:00.99  CPU: 14%   major: 0    minor: 197    invol: 8    vol: 281
TIME: 0:00.09  CPU: 100%  major: 0    minor: 197    invol: 1    vol: 1

TIME: 0:00.98  CPU: 15%   major: 0    minor: 198    invol: 8    vol: 272
TIME: 0:00.09  CPU: 101%  major: 0    minor: 198    invol: 1    vol: 1

101
-------------------------------------------------------------------------------

time ctls c_100.ct:

TIME: 0:00.02  CPU: 0%    major: 0    minor: 122    invol: 1    vol: 11
TIME: 0:00.00  CPU: 0%    major: 0    minor: 121    invol: 1    vol: 2

TIME: 0:00.01  CPU: 36%   major: 0    minor: 123    invol: 0    vol: 7
TIME: 0:00.00  CPU: 0%    major: 0    minor: 122    invol: 1    vol: 2

TIME: 0:00.00  CPU: 0%    major: 0    minor: 122    invol: 0    vol: 7
TIME: 0:00.00  CPU: 0%    major: 0    minor: 121    invol: 1    vol: 2

TIME: 0:00.00  CPU: 200%  major: 0    minor: 121    invol: 1    vol: 7
TIME: 0:00.00  CPU: 0%    major: 0    minor: 121    invol: 1    vol: 2

TIME: 0:00.00  CPU: 0%    major: 0    minor: 123    invol: 1    vol: 7
TIME: 0:00.00  CPU: 0%    major: 0    minor: 121    invol: 1    vol: 2

100
-------------------------------------------------------------------------------

time ls -l V_100:

TIME: 0:00.02  CPU: 44%   major: 0    minor: 243    invol: 4    vol: 9
TIME: 0:00.00  CPU: 133%  major: 0    minor: 244    invol: 4    vol: 3

TIME: 0:00.00  CPU: 100%  major: 0    minor: 244    invol: 3    vol: 9
TIME: 0:00.00  CPU: 133%  major: 0    minor: 244    invol: 4    vol: 3

TIME: 0:00.00  CPU: 100%  major: 0    minor: 243    invol: 5    vol: 7
TIME: 0:00.00  CPU: 133%  major: 0    minor: 245    invol: 4    vol: 3
```

```
11
-------------------------------------------------------------------------------

time ctls -l c_10.ct:

TIME: 0:00.00  CPU: 0%    major: 0    minor: 150    invol: 1    vol: 10
TIME: 0:00.00  CPU: 0%    major: 0    minor: 148    invol: 3    vol: 4

TIME: 0:00.00  CPU: 0%    major: 0    minor: 150    invol: 1    vol: 6
TIME: 0:00.00  CPU: 400%  major: 0    minor: 150    invol: 3    vol: 4

TIME: 0:00.00  CPU: 800%  major: 0    minor: 150    invol: 1    vol: 6
TIME: 0:00.00  CPU: 400%  major: 0    minor: 148    invol: 1    vol: 6

TIME: 0:00.00  CPU: 400%  major: 0    minor: 149    invol: 3    vol: 4
TIME: 0:00.00  CPU: 400%  major: 0    minor: 150    invol: 1    vol: 6

TIME: 0:00.00  CPU: 400%  major: 0    minor: 149    invol: 1    vol: 6
TIME: 0:00.00  CPU: 400%  major: 0    minor: 149    invol: 3    vol: 4

10
-------------------------------------------------------------------------------

/dev/md2 on /raid/heinrich type reiserfs (rw,noexec,nosuid,nodev)
100 (256*4 KB) Dateien on reiserfs
-------------------------------------------------------------------------------

time create_testfiles2.sh V_100:
TIME: 0:02.15  CPU: 66%   major: 0    minor: 22311    invol: 210    vol: 211

time ctmk c_100.ct:
TIME: 0:01.88  CPU: 57%   major: 0    minor: 135    invol: 116    vol: 481

time tar -cvf V_100:
tar: Removing leading `/' from member names
TIME: 0:02.47  CPU: 25%   major: 0    minor: 219    invol: 39    vol: 496
-------------------------------------------------------------------------------

time ls -l V_100:

TIME: 0:00.00  CPU: 0%    major: 0    minor: 223    invol: 3    vol: 3
TIME: 0:00.00  CPU: 200%  major: 0    minor: 224    invol: 1    vol: 5

TIME: 0:00.00  CPU: 200%  major: 0    minor: 224    invol: 1    vol: 5
TIME: 0:00.00  CPU: 0%    major: 0    minor: 224    invol: 3    vol: 3

TIME: 0:00.00  CPU: 200%  major: 0    minor: 224    invol: 4    vol: 1
TIME: 0:00.00  CPU: 200%  major: 0    minor: 223    invol: 4    vol: 3

TIME: 0:00.00  CPU: 200%  major: 0    minor: 223    invol: 1    vol: 5
TIME: 0:00.00  CPU: 0%    major: 0    minor: 223    invol: 3    vol: 3

TIME: 0:00.00  CPU: 200%  major: 0    minor: 224    invol: 4    vol: 1
TIME: 0:00.00  CPU: 0%    major: 0    minor: 224    invol: 0    vol: 4

100
-------------------------------------------------------------------------------

time tar -tf T_100.tar:

TIME: 0:00.99  CPU: 18%   major: 0    minor: 198    invol: 20    vol: 370
TIME: 0:00.09  CPU: 101%  major: 0    minor: 198    invol: 0    vol: 1

TIME: 0:01.07  CPU: 16%   major: 0    minor: 198    invol: 24    vol: 348
TIME: 0:00.09  CPU: 101%  major: 0    minor: 197    invol: 5    vol: 1

TIME: 0:00.98  CPU: 20%   major: 0    minor: 197    invol: 29    vol: 362
TIME: 0:00.09  CPU: 101%  major: 0    minor: 196    invol: 2    vol: 1

TIME: 0:01.11  CPU: 17%   major: 0    minor: 197    invol: 25    vol: 347
TIME: 0:00.09  CPU: 101%  major: 0    minor: 198    invol: 1    vol: 1

TIME: 0:00.99  CPU: 20%   major: 0    minor: 197    invol: 25    vol: 341
TIME: 0:00.09  CPU: 101%  major: 0    minor: 197    invol: 0    vol: 1

101
-------------------------------------------------------------------------------

time ctls c_100.ct:

TIME: 0:00.01  CPU: 0%    major: 0    minor: 121    invol: 0    vol: 8
TIME: 0:00.00  CPU: 0%    major: 0    minor: 121    invol: 0    vol: 2

TIME: 0:00.00  CPU: 0%    major: 0    minor: 121    invol: 0    vol: 6
TIME: 0:00.00  CPU: 0%    major: 0    minor: 121    invol: 0    vol: 2

TIME: 0:00.00  CPU: 57%   major: 0    minor: 121    invol: 1    vol: 6
TIME: 0:00.00  CPU: 0%    major: 0    minor: 121    invol: 0    vol: 2

TIME: 0:00.00  CPU: 0%    major: 0    minor: 122    invol: 1    vol: 6
TIME: 0:00.00  CPU: 0%    major: 0    minor: 121    invol: 0    vol: 2

TIME: 0:00.00  CPU: 200%  major: 0    minor: 120    invol: 1    vol: 6
TIME: 0:00.00  CPU: 0%    major: 0    minor: 122    invol: 0    vol: 2

100
-------------------------------------------------------------------------------

time ls -l V_100:

TIME: 0:00.03  CPU: 22%   major: 0    minor: 242    invol: 5    vol: 29
TIME: 0:00.00  CPU: 66%   major: 0    minor: 243    invol: 0    vol: 4

TIME: 0:00.01  CPU: 53%   major: 0    minor: 242    invol: 4    vol: 31
TIME: 0:00.00  CPU: 66%   major: 0    minor: 243    invol: 0    vol: 4

TIME: 0:00.02  CPU: 36%   major: 0    minor: 242    invol: 5    vol: 29
TIME: 0:00.00  CPU: 66%   major: 0    minor: 241    invol: 2    vol: 2
```

```
TIME: 0:00.00  CPU: 100%  major: 0  minor: 243  invol: 2  vol: 9
TIME: 0:00.00  CPU: 66%  major: 0  minor: 243  invol: 0  vol: 4

TIME: 0:00.00  CPU: 100%  major: 0  minor: 244  invol: 0  vol: 11
TIME: 0:00.00  CPU: 133%  major: 0  minor: 244  invol: 4  vol: 3

101
-------------------------------------------------------------------------------

time tar -tvf T_100.tar:

TIME: 0:01.02  CPU: 17%  major: 0  minor: 208  invol: 5  vol: 259
TIME: 0:00.09  CPU: 97%  major: 0  minor: 207  invol: 1  vol: 1

TIME: 0:01.00  CPU: 13%  major: 0  minor: 209  invol: 9  vol: 289
TIME: 0:00.09  CPU: 97%  major: 0  minor: 209  invol: 1  vol: 1

TIME: 0:01.00  CPU: 14%  major: 0  minor: 209  invol: 6  vol: 260
TIME: 0:00.09  CPU: 97%  major: 0  minor: 208  invol: 2  vol: 1

TIME: 0:01.00  CPU: 15%  major: 0  minor: 208  invol: 6  vol: 259
TIME: 0:00.09  CPU: 97%  major: 0  minor: 208  invol: 0  vol: 1

TIME: 0:01.00  CPU: 15%  major: 0  minor: 209  invol: 4  vol: 257
TIME: 0:00.09  CPU: 97%  major: 0  minor: 208  invol: 2  vol: 1

101
-------------------------------------------------------------------------------

time ctls -l c_100.ct:

TIME: 0:00.02  CPU: 0%  major: 0  minor: 151  invol: 5  vol: 7
TIME: 0:00.00  CPU: 0%  major: 0  minor: 149  invol: 4  vol: 4

TIME: 0:00.00  CPU: 400%  major: 0  minor: 151  invol: 1  vol: 6
TIME: 0:00.00  CPU: 400%  major: 0  minor: 151  invol: 5  vol: 2

TIME: 0:00.00  CPU: 400%  major: 0  minor: 150  invol: 4  vol: 4
TIME: 0:00.00  CPU: 400%  major: 0  minor: 151  invol: 1  vol: 6

TIME: 0:00.00  CPU: 400%  major: 0  minor: 151  invol: 5  vol: 2
TIME: 0:00.00  CPU: 400%  major: 0  minor: 151  invol: 4  vol: 4

TIME: 0:00.00  CPU: 400%  major: 0  minor: 152  invol: 1  vol: 6
TIME: 0:00.00  CPU: 400%  major: 0  minor: 150  invol: 5  vol: 2

100
-------------------------------------------------------------------------------

/dev/md2 on /raid/heinrich type ext3 (rw,noexec,nosuid,nodev)
1000 (256*4 KB) Dateien on ext3
-------------------------------------------------------------------------------

time create_testfiles2.sh V_1000:
TIME: 0:12.31  CPU: 82%  major: 0  minor: 216625  invol: 2220  vol: 2011

time ctmk c_1000.ct:
TIME: 0:47.10  CPU: 22%  major: 0  minor: 183  invol: 1759  vol: 5776

time tar -cvf V_1000:
tar: Removing leading `/' from member names
TIME: 0:50.06  CPU: 16%  major: 3  minor: 220  invol: 1396  vol: 5898
-------------------------------------------------------------------------------

time ls -l V_1000:

TIME: 0:00.08  CPU: 9%  major: 2  minor: 304  invol: 1  vol: 24
TIME: 0:00.00  CPU: 150%  major: 0  minor: 306  invol: 5  vol: 1

TIME: 0:00.01  CPU: 80%  major: 0  minor: 305  invol: 3  vol: 16
TIME: 0:00.00  CPU: 150%  major: 0  minor: 306  invol: 5  vol: 1

TIME: 0:00.01  CPU: 120%  major: 0  minor: 305  invol: 1  vol: 16
TIME: 0:00.00  CPU: 150%  major: 0  minor: 306  invol: 4  vol: 3

TIME: 0:00.01  CPU: 120%  major: 0  minor: 306  invol: 5  vol: 12
TIME: 0:00.00  CPU: 200%  major: 0  minor: 306  invol: 4  vol: 5

TIME: 0:00.00  CPU: 133%  major: 0  minor: 304  invol: 3  vol: 14
TIME: 0:00.00  CPU: 150%  major: 0  minor: 305  invol: 4  vol: 3

1000
-------------------------------------------------------------------------------

time tar -tf T_1000.tar:

TIME: 0:09.76  CPU: 13%  major: 1  minor: 196  invol: 92  vol: 2540
TIME: 0:09.09  CPU: 15%  major: 0  minor: 198  invol: 566  vol: 4117

TIME: 0:09.82  CPU: 13%  major: 0  minor: 196  invol: 83  vol: 2533
TIME: 0:08.94  CPU: 15%  major: 0  minor: 198  invol: 569  vol: 4204

TIME: 0:09.59  CPU: 13%  major: 0  minor: 198  invol: 103  vol: 2829
TIME: 0:08.95  CPU: 16%  major: 0  minor: 197  invol: 585  vol: 4333

TIME: 0:09.65  CPU: 14%  major: 0  minor: 198  invol: 110  vol: 2739
TIME: 0:08.98  CPU: 15%  major: 0  minor: 197  invol: 543  vol: 4109

TIME: 0:09.77  CPU: 14%  major: 0  minor: 198  invol: 74  vol: 2515
TIME: 0:09.10  CPU: 16%  major: 0  minor: 197  invol: 518  vol: 4031

1001
-------------------------------------------------------------------------------

time ctls c_1000.ct:

TIME: 0:00.03  CPU: 10%  major: 0  minor: 125  invol: 1  vol: 16
TIME: 0:00.00  CPU: 0%  major: 0  minor: 125  invol: 0  vol: 2
```

```
TIME: 0:00.02  CPU: 72%  major: 0  minor: 243  invol: 5  vol: 29
TIME: 0:00.00  CPU: 66%  major: 0  minor: 241  invol: 2  vol: 3

TIME: 0:00.02  CPU: 38%  major: 0  minor: 243  invol: 5  vol: 29
TIME: 0:00.00  CPU: 66%  major: 0  minor: 243  invol: 2  vol: 2

101
-------------------------------------------------------------------------------

time tar -tvf T_100.tar:

TIME: 0:01.03  CPU: 19%  major: 0  minor: 210  invol: 23  vol: 336
TIME: 0:00.09  CPU: 97%  major: 0  minor: 207  invol: 2  vol: 1

TIME: 0:01.04  CPU: 19%  major: 0  minor: 208  invol: 23  vol: 355
TIME: 0:00.09  CPU: 97%  major: 0  minor: 210  invol: 1  vol: 1

TIME: 0:01.02  CPU: 18%  major: 0  minor: 209  invol: 33  vol: 314
TIME: 0:00.09  CPU: 97%  major: 0  minor: 210  invol: 1  vol: 1

TIME: 0:01.06  CPU: 18%  major: 0  minor: 209  invol: 19  vol: 333
TIME: 0:00.09  CPU: 97%  major: 0  minor: 209  invol: 1  vol: 1

TIME: 0:01.03  CPU: 21%  major: 0  minor: 209  invol: 34  vol: 355
TIME: 0:00.09  CPU: 98%  major: 0  minor: 209  invol: 1  vol: 1

101
-------------------------------------------------------------------------------

time ctls -l c_100.ct:

TIME: 0:00.04  CPU: 0%  major: 0  minor: 150  invol: 4  vol: 6
TIME: 0:00.00  CPU: 0%  major: 0  minor: 152  invol: 2  vol: 4

TIME: 0:00.00  CPU: 0%  major: 0  minor: 150  invol: 4  vol: 4
TIME: 0:00.00  CPU: 0%  major: 0  minor: 150  invol: 4  vol: 2

TIME: 0:00.00  CPU: 0%  major: 0  minor: 152  invol: 3  vol: 4
TIME: 0:00.00  CPU: 0%  major: 0  minor: 151  invol: 4  vol: 4

TIME: 0:00.00  CPU: 0%  major: 0  minor: 151  invol: 1  vol: 6
TIME: 0:00.00  CPU: 0%  major: 0  minor: 150  invol: 3  vol: 4

TIME: 0:00.00  CPU: 400%  major: 0  minor: 151  invol: 4  vol: 4
TIME: 0:00.00  CPU: 0%  major: 0  minor: 151  invol: 1  vol: 6

100
-------------------------------------------------------------------------------

/dev/md2 on /raid/heinrich type reiserfs (rw,noexec,nosuid,nodev)
1000 (256*4 KB) Dateien on reiserfs
-------------------------------------------------------------------------------

time create_testfiles2.sh V_1000:
TIME: 0:12.69  CPU: 93%  major: 0  minor: 216614  invol: 2279  vol: 1989

time ctmk c_1000.ct:
TIME: 0:31.87  CPU: 45%  major: 0  minor: 184  invol: 2507  vol: 5038

time tar -cvf V_1000:
tar: Removing leading `/' from member names
TIME: 0:29.69  CPU: 26%  major: 3  minor: 220  invol: 1791  vol: 4909
-------------------------------------------------------------------------------

time ls -l V_1000:

TIME: 0:00.04  CPU: 27%  major: 2  minor: 290  invol: 1  vol: 13
TIME: 0:00.00  CPU: 133%  major: 0  minor: 292  invol: 2  vol: 5

TIME: 0:00.00  CPU: 133%  major: 0  minor: 291  invol: 3  vol: 5
TIME: 0:00.00  CPU: 133%  major: 0  minor: 291  invol: 3  vol: 2

TIME: 0:00.00  CPU: 133%  major: 0  minor: 292  invol: 4  vol: 3
TIME: 0:00.00  CPU: 133%  major: 0  minor: 292  invol: 5  vol: 1

TIME: 0:00.00  CPU: 133%  major: 0  minor: 293  invol: 2  vol: 4
TIME: 0:00.00  CPU: 66%  major: 0  minor: 291  invol: 1  vol: 5

TIME: 0:00.00  CPU: 133%  major: 0  minor: 292  invol: 4  vol: 5
TIME: 0:00.00  CPU: 66%  major: 0  minor: 291  invol: 3  vol: 3

1000
-------------------------------------------------------------------------------

time tar -tf T_1000.tar:

TIME: 0:10.23  CPU: 19%  major: 1  minor: 196  invol: 206  vol: 3178
TIME: 0:09.08  CPU: 22%  major: 0  minor: 199  invol: 686  vol: 4473

TIME: 0:10.41  CPU: 18%  major: 0  minor: 199  invol: 198  vol: 2963
TIME: 0:08.96  CPU: 23%  major: 0  minor: 198  invol: 631  vol: 4465

TIME: 0:10.34  CPU: 19%  major: 0  minor: 197  invol: 245  vol: 3051
TIME: 0:09.00  CPU: 21%  major: 0  minor: 197  invol: 521  vol: 4498

TIME: 0:10.39  CPU: 18%  major: 0  minor: 199  invol: 212  vol: 3142
TIME: 0:09.15  CPU: 23%  major: 0  minor: 197  invol: 582  vol: 4244

TIME: 0:10.39  CPU: 19%  major: 0  minor: 197  invol: 175  vol: 2984
TIME: 0:09.10  CPU: 21%  major: 0  minor: 198  invol: 575  vol: 4222

1001
-------------------------------------------------------------------------------

time ctls c_1000.ct:

TIME: 0:00.05  CPU: 0%  major: 0  minor: 125  invol: 1  vol: 15
TIME: 0:00.00  CPU: 0%  major: 0  minor: 125  invol: 1  vol: 2
```

TIME: 0:00.00 CPU: 200% major: 0 minor: 126 invol: 1 vol: 8
TIME: 0:00.00 CPU: 400% major: 0 minor: 125 invol: 1 vol: 2

TIME: 0:00.00 CPU: 200% major: 0 minor: 125 invol: 1 vol: 8
TIME: 0:00.00 CPU: 400% major: 0 minor: 126 invol: 1 vol: 2

TIME: 0:00.00 CPU: 200% major: 0 minor: 125 invol: 1 vol: 8
TIME: 0:00.00 CPU: 400% major: 0 minor: 124 invol: 1 vol: 2

TIME: 0:00.00 CPU: 200% major: 0 minor: 125 invol: 1 vol: 8
TIME: 0:00.00 CPU: 400% major: 0 minor: 127 invol: 1 vol: 2

1000

---

time ls -l V_1000:

TIME: 0:00.22 CPU: 24% major: 2 minor: 322 invol: 3 vol: 54
TIME: 0:00.04 CPU: 97% major: 0 minor: 324 invol: 4 vol: 2

TIME: 0:00.18 CPU: 29% major: 0 minor: 324 invol: 5 vol: 45
TIME: 0:00.04 CPU: 95% major: 0 minor: 324 invol: 3 vol: 3

TIME: 0:00.19 CPU: 30% major: 0 minor: 325 invol: 3 vol: 45
TIME: 0:00.04 CPU: 95% major: 0 minor: 324 invol: 3 vol: 3

TIME: 0:00.17 CPU: 36% major: 0 minor: 324 invol: 2 vol: 45
TIME: 0:00.04 CPU: 97% major: 0 minor: 324 invol: 3 vol: 3

TIME: 0:00.11 CPU: 50% major: 0 minor: 325 invol: 5 vol: 45
TIME: 0:00.04 CPU: 107% major: 0 minor: 324 invol: 3 vol: 3

1001

---

time tar -tvf T_1000.tar:

TIME: 0:09.70 CPU: 14% major: 0 minor: 208 invol: 98 vol: 2704
TIME: 0:08.86 CPU: 15% major: 0 minor: 208 invol: 494 vol: 4388

TIME: 0:09.72 CPU: 13% major: 0 minor: 210 invol: 89 vol: 2636
TIME: 0:09.16 CPU: 15% major: 0 minor: 209 invol: 587 vol: 3877

TIME: 0:09.77 CPU: 14% major: 0 minor: 210 invol: 126 vol: 2717
TIME: 0:08.84 CPU: 15% major: 0 minor: 208 invol: 518 vol: 4469

TIME: 0:09.80 CPU: 14% major: 0 minor: 209 invol: 88 vol: 2481
TIME: 0:09.04 CPU: 15% major: 0 minor: 209 invol: 507 vol: 3985

TIME: 0:09.72 CPU: 13% major: 0 minor: 208 invol: 117 vol: 2690
TIME: 0:08.97 CPU: 15% major: 0 minor: 209 invol: 528 vol: 4261

1001

---

time ctls -l c_1000.ct:

TIME: 0:00.03 CPU: 10% major: 0 minor: 157 invol: 3 vol: 20
TIME: 0:00.00 CPU: 133% major: 0 minor: 157 invol: 3 vol: 4

TIME: 0:00.00 CPU: 133% major: 0 minor: 157 invol: 4 vol: 4
TIME: 0:00.00 CPU: 0% major: 0 minor: 158 invol: 3 vol: 4

TIME: 0:00.00 CPU: 133% major: 0 minor: 157 invol: 4 vol: 4
TIME: 0:00.00 CPU: 133% major: 0 minor: 157 invol: 4 vol: 4

TIME: 0:00.00 CPU: 133% major: 0 minor: 157 invol: 3 vol: 4
TIME: 0:00.00 CPU: 0% major: 0 minor: 158 invol: 4 vol: 4

TIME: 0:00.00 CPU: 133% major: 0 minor: 157 invol: 4 vol: 4
TIME: 0:00.00 CPU: 0% major: 0 minor: 157 invol: 3 vol: 4

1000

/dev/md2 on /raid/heinrich type ext3 (rw,noexec,nosuid,nodev)
10000 (256*4 KB) Dateien on ext3

---

time create_testfiles2.sh V_10000:
TIME: 2:26.10 CPU: 71% major: 7 minor: 2139502 invol: 26269 vol: 21589

time ctmk c_10000.ct:
TIME: 5:56.05 CPU: 31% major: 0 minor: 498 invol: 20827 vol: 56255

time tar -cvf V_10000:
tar: Removing leading `/' from member names
TIME: 8:29.17 CPU: 16% major: 3 minor: 272 invol: 18091 vol: 56463

---

time ls -l V_10000:

TIME: 0:00.75 CPU: 11% major: 4 minor: 859 invol: 6 vol: 106
TIME: 0:00.08 CPU: 101% major: 0 minor: 862 invol: 3 vol: 2

TIME: 0:00.68 CPU: 16% major: 0 minor: 863 invol: 0 vol: 90
TIME: 0:00.08 CPU: 97% major: 0 minor: 861 invol: 3 vol: 2

TIME: 0:00.69 CPU: 12% major: 0 minor: 864 invol: 0 vol: 90
TIME: 0:00.08 CPU: 96% major: 0 minor: 862 invol: 6 vol: 2

TIME: 0:00.74 CPU: 11% major: 0 minor: 863 invol: 0 vol: 90
TIME: 0:00.08 CPU: 97% major: 0 minor: 862 invol: 6 vol: 2

TIME: 0:00.66 CPU: 11% major: 0 minor: 862 invol: 0 vol: 90
TIME: 0:00.08 CPU: 98% major: 0 minor: 862 invol: 3 vol: 2

10000

TIME: 0:00.00 CPU: 200% major: 0 minor: 126 invol: 1 vol: 7
TIME: 0:00.00 CPU: 400% major: 0 minor: 125 invol: 1 vol: 2

TIME: 0:00.02 CPU: 0% major: 0 minor: 124 invol: 1 vol: 7
TIME: 0:00.00 CPU: 0% major: 0 minor: 126 invol: 1 vol: 2

TIME: 0:00.00 CPU: 50% major: 0 minor: 127 invol: 1 vol: 7
TIME: 0:00.00 CPU: 0% major: 0 minor: 126 invol: 1 vol: 2

TIME: 0:00.01 CPU: 0% major: 0 minor: 124 invol: 0 vol: 7
TIME: 0:00.00 CPU: 0% major: 0 minor: 126 invol: 1 vol: 2

1000

---

time ls -l V_1000:

TIME: 0:01.47 CPU: 4% major: 2 minor: 309 invol: 5 vol: 291
TIME: 0:00.04 CPU: 100% major: 0 minor: 310 invol: 2 vol: 4

TIME: 0:01.43 CPU: 4% major: 0 minor: 309 invol: 0 vol: 282
TIME: 0:00.04 CPU: 100% major: 0 minor: 312 invol: 3 vol: 3

TIME: 0:01.45 CPU: 3% major: 0 minor: 312 invol: 2 vol: 281
TIME: 0:00.04 CPU: 100% major: 0 minor: 309 invol: 5 vol: 1

TIME: 0:01.42 CPU: 3% major: 0 minor: 309 invol: 4 vol: 282
TIME: 0:00.04 CPU: 100% major: 0 minor: 311 invol: 7 vol: 3

TIME: 0:01.36 CPU: 3% major: 0 minor: 310 invol: 2 vol: 284
TIME: 0:00.04 CPU: 100% major: 0 minor: 311 invol: 5 vol: 1

1001

---

time tar -tvf T_1000.tar:

TIME: 0:10.29 CPU: 19% major: 0 minor: 208 invol: 215 vol: 3002
TIME: 0:09.09 CPU: 23% major: 0 minor: 208 invol: 600 vol: 4373

TIME: 0:10.15 CPU: 18% major: 0 minor: 208 invol: 191 vol: 3196
TIME: 0:08.84 CPU: 24% major: 0 minor: 209 invol: 596 vol: 4470

TIME: 0:10.13 CPU: 19% major: 0 minor: 210 invol: 208 vol: 3185
TIME: 0:09.05 CPU: 23% major: 0 minor: 208 invol: 589 vol: 4225

TIME: 0:10.26 CPU: 18% major: 0 minor: 208 invol: 178 vol: 3072
TIME: 0:09.07 CPU: 23% major: 0 minor: 208 invol: 593 vol: 4397

TIME: 0:10.28 CPU: 18% major: 0 minor: 208 invol: 202 vol: 3018
TIME: 0:09.35 CPU: 22% major: 0 minor: 208 invol: 510 vol: 3917

1001

---

time ctls -l c_1000.ct:

TIME: 0:00.06 CPU: 13% major: 0 minor: 157 invol: 2 vol: 21
TIME: 0:00.00 CPU: 0% major: 0 minor: 158 invol: 4 vol: 4

TIME: 0:00.00 CPU: 133% major: 0 minor: 157 invol: 1 vol: 5
TIME: 0:00.00 CPU: 266% major: 0 minor: 158 invol: 5 vol: 2

TIME: 0:00.00 CPU: 133% major: 0 minor: 158 invol: 5 vol: 2
TIME: 0:00.00 CPU: 133% major: 0 minor: 158 invol: 3 vol: 6

TIME: 0:00.00 CPU: 133% major: 0 minor: 157 invol: 3 vol: 4
TIME: 0:00.00 CPU: 133% major: 0 minor: 157 invol: 3 vol: 4

TIME: 0:00.00 CPU: 133% major: 0 minor: 157 invol: 5 vol: 2
TIME: 0:00.00 CPU: 0% major: 0 minor: 158 invol: 3 vol: 6

1000

/dev/md2 on /raid/heinrich type reiserfs (rw,noexec,nosuid,nodev)
10000 (256*4 KB) Dateien on reiserfs

---

time create_testfiles2.sh V_10000:
TIME: 2:14.64 CPU: 91% major: 7 minor: 2139440 invol: 25261 vol: 20064

time ctmk c_10000.ct:
TIME: 5:54.71 CPU: 41% major: 0 minor: 469 invol: 26017 vol: 51241

time tar -cvf V_10000:
tar: Removing leading `/' from member names
TIME: 5:19.62 CPU: 26% major: 4 minor: 272 invol: 14194 vol: 49700

---

time ls -l V_10000:

TIME: 0:00.15 CPU: 31% major: 4 minor: 713 invol: 5 vol: 27
TIME: 0:00.04 CPU: 97% major: 0 minor: 717 invol: 1 vol: 5

TIME: 0:00.07 CPU: 59% major: 0 minor: 715 invol: 1 vol: 12
TIME: 0:00.04 CPU: 97% major: 0 minor: 716 invol: 3 vol: 3

TIME: 0:00.09 CPU: 57% major: 0 minor: 716 invol: 4 vol: 8
TIME: 0:00.04 CPU: 97% major: 0 minor: 715 invol: 3 vol: 3

TIME: 0:00.07 CPU: 66% major: 0 minor: 717 invol: 1 vol: 12
TIME: 0:00.04 CPU: 97% major: 0 minor: 717 invol: 3 vol: 2

TIME: 0:00.08 CPU: 55% major: 0 minor: 718 invol: 3 vol: 10
TIME: 0:00.04 CPU: 97% major: 0 minor: 716 invol: 1 vol: 5

10000

---

time tar -tf T_10000.tar:

TIME: 1:37.19 CPU: 14% major: 2 minor: 196 invol: 1969 vol: 27670
TIME: 1:37.36 CPU: 14% major: 3 minor: 195 invol: 2142 vol: 28020

TIME: 1:37.63 CPU: 14% major: 3 minor: 195 invol: 1555 vol: 27499
TIME: 1:37.98 CPU: 15% major: 3 minor: 193 invol: 1758 vol: 27582

TIME: 1:37.57 CPU: 14% major: 3 minor: 195 invol: 1403 vol: 27272
TIME: 1:37.11 CPU: 14% major: 3 minor: 195 invol: 2103 vol: 28256

TIME: 1:37.34 CPU: 14% major: 3 minor: 194 invol: 2121 vol: 27646
TIME: 1:38.02 CPU: 14% major: 3 minor: 194 invol: 1535 vol: 27203

TIME: 1:37.83 CPU: 14% major: 3 minor: 194 invol: 1856 vol: 27379
TIME: 1:37.67 CPU: 15% major: 3 minor: 194 invol: 2093 vol: 28076

10001

---

time ctls c_10000.ct:

TIME: 0:00.06 CPU: 6% major: 0 minor: 164 invol: 1 vol: 22
TIME: 0:00.00 CPU: 133% major: 0 minor: 163 invol: 1 vol: 2

TIME: 0:00.03 CPU: 12% major: 0 minor: 164 invol: 1 vol: 10
TIME: 0:00.00 CPU: 0% major: 0 minor: 164 invol: 1 vol: 2

TIME: 0:00.02 CPU: 0% major: 0 minor: 164 invol: 1 vol: 10
TIME: 0:00.00 CPU: 0% major: 0 minor: 163 invol: 1 vol: 2

TIME: 0:00.02 CPU: 20% major: 0 minor: 164 invol: 1 vol: 10
TIME: 0:00.00 CPU: 0% major: 0 minor: 162 invol: 1 vol: 2

TIME: 0:00.02 CPU: 20% major: 0 minor: 164 invol: 0 vol: 10
TIME: 0:00.00 CPU: 133% major: 0 minor: 163 invol: 0 vol: 2

10000

---

time ls -l V_10000:

TIME: 0:01.64 CPU: 35% major: 4 minor: 878 invol: 9 vol: 421
TIME: 0:00.41 CPU: 100% major: 0 minor: 881 invol: 6 vol: 2

TIME: 0:01.56 CPU: 34% major: 0 minor: 882 invol: 1 vol: 404
TIME: 0:00.41 CPU: 100% major: 0 minor: 881 invol: 4 vol: 5

TIME: 0:01.61 CPU: 38% major: 0 minor: 880 invol: 6 vol: 404
TIME: 0:00.43 CPU: 100% major: 0 minor: 881 invol: 6 vol: 1

TIME: 0:01.53 CPU: 37% major: 0 minor: 880 invol: 0 vol: 403
TIME: 0:00.41 CPU: 100% major: 0 minor: 882 invol: 5 vol: 3

TIME: 0:01.59 CPU: 36% major: 0 minor: 880 invol: 3 vol: 401
TIME: 0:00.41 CPU: 99% major: 0 minor: 881 invol: 8 vol: 1

10001

---

time tar -tvf T_10000.tar:

TIME: 1:36.91 CPU: 14% major: 1 minor: 208 invol: 1722 vol: 27938
TIME: 1:37.76 CPU: 15% major: 3 minor: 206 invol: 1940 vol: 27975

TIME: 1:37.74 CPU: 14% major: 3 minor: 205 invol: 1916 vol: 27717
TIME: 1:37.91 CPU: 15% major: 3 minor: 206 invol: 1957 vol: 28127

TIME: 1:37.37 CPU: 14% major: 3 minor: 205 invol: 2400 vol: 27954
TIME: 1:37.75 CPU: 15% major: 3 minor: 206 invol: 2045 vol: 27779

TIME: 1:37.33 CPU: 14% major: 3 minor: 205 invol: 2084 vol: 28106
TIME: 1:38.01 CPU: 15% major: 3 minor: 206 invol: 2213 vol: 28248

TIME: 1:37.79 CPU: 14% major: 3 minor: 205 invol: 1813 vol: 27808
TIME: 1:37.90 CPU: 14% major: 3 minor: 205 invol: 2287 vol: 28533

10001

---

time ctls -l c_10000.ct:

TIME: 0:00.09 CPU: 24% major: 0 minor: 232 invol: 4 vol: 29
TIME: 0:00.02 CPU: 100% major: 0 minor: 233 invol: 3 vol: 4

TIME: 0:00.01 CPU: 105% major: 0 minor: 231 invol: 1 vol: 6
TIME: 0:00.02 CPU: 100% major: 0 minor: 231 invol: 1 vol: 6

TIME: 0:00.02 CPU: 100% major: 0 minor: 231 invol: 3 vol: 4
TIME: 0:00.01 CPU: 105% major: 0 minor: 232 invol: 1 vol: 6

TIME: 0:00.02 CPU: 100% major: 0 minor: 231 invol: 3 vol: 6
TIME: 0:00.02 CPU: 96% major: 0 minor: 231 invol: 3 vol: 4

TIME: 0:00.02 CPU: 95% major: 0 minor: 230 invol: 1 vol: 6
TIME: 0:00.02 CPU: 100% major: 0 minor: 231 invol: 1 vol: 6

10000

/dev/md2 on /raid/heinrich type ext3 (rw,noexec,nosuid,nodev)
100 (256*4 KB) Dateien on ext3

---

time create_testfiles2.sh V_100:
TIME: 0:01.43 CPU: 90% major: 7 minor: 22097 invol: 212 vol: 231

---

time tar -tf T_10000.tar:

TIME: 1:40.63 CPU: 18% major: 2 minor: 195 invol: 6398 vol: 34240
TIME: 1:40.31 CPU: 18% major: 3 minor: 194 invol: 7048 vol: 35106

TIME: 1:40.80 CPU: 18% major: 3 minor: 194 invol: 6543 vol: 34441
TIME: 1:40.00 CPU: 18% major: 3 minor: 194 invol: 7096 vol: 35468

TIME: 1:40.40 CPU: 19% major: 3 minor: 195 invol: 6384 vol: 34443
TIME: 1:40.14 CPU: 18% major: 3 minor: 195 invol: 7067 vol: 35279

TIME: 1:40.68 CPU: 18% major: 3 minor: 195 invol: 6404 vol: 34131
TIME: 1:40.03 CPU: 18% major: 3 minor: 195 invol: 6923 vol: 35307

TIME: 1:40.30 CPU: 18% major: 3 minor: 194 invol: 6505 vol: 34196
TIME: 1:40.12 CPU: 19% major: 3 minor: 194 invol: 6953 vol: 35010

10001

---

time ctls c_10000.ct:

TIME: 0:00.05 CPU: 7% major: 0 minor: 163 invol: 1 vol: 20
TIME: 0:00.00 CPU: 133% major: 0 minor: 164 invol: 1 vol: 2

TIME: 0:00.02 CPU: 20% major: 0 minor: 164 invol: 1 vol: 7
TIME: 0:00.00 CPU: 133% major: 0 minor: 163 invol: 1 vol: 2

TIME: 0:00.00 CPU: 133% major: 0 minor: 163 invol: 1 vol: 7
TIME: 0:00.00 CPU: 133% major: 0 minor: 163 invol: 1 vol: 2

TIME: 0:00.02 CPU: 0% major: 0 minor: 164 invol: 1 vol: 7
TIME: 0:00.00 CPU: 133% major: 0 minor: 163 invol: 1 vol: 2

TIME: 0:00.02 CPU: 0% major: 0 minor: 164 invol: 1 vol: 8
TIME: 0:00.00 CPU: 133% major: 0 minor: 163 invol: 1 vol: 2

10000

---

time ls -l V_10000:

TIME: 0:17.07 CPU: 3% major: 4 minor: 730 invol: 8 vol: 2803
TIME: 0:00.38 CPU: 99% major: 0 minor: 735 invol: 8 vol: 1

TIME: 0:16.49 CPU: 4% major: 0 minor: 735 invol: 10 vol: 2783
TIME: 0:00.38 CPU: 99% major: 0 minor: 735 invol: 2 vol: 2

TIME: 0:16.64 CPU: 3% major: 0 minor: 735 invol: 3 vol: 2785
TIME: 0:00.42 CPU: 95% major: 0 minor: 734 invol: 2 vol: 5

TIME: 0:16.38 CPU: 3% major: 0 minor: 735 invol: 15 vol: 2787
TIME: 0:00.38 CPU: 100% major: 0 minor: 736 invol: 6 vol: 4

TIME: 0:16.58 CPU: 3% major: 0 minor: 736 invol: 20 vol: 2784
TIME: 0:00.38 CPU: 100% major: 0 minor: 735 invol: 4 vol: 4

10001

---

time tar -tvf T_10000.tar:

TIME: 1:40.31 CPU: 19% major: 1 minor: 208 invol: 6391 vol: 34119
TIME: 1:40.06 CPU: 19% major: 3 minor: 206 invol: 7054 vol: 35108

TIME: 1:40.26 CPU: 18% major: 3 minor: 205 invol: 6523 vol: 34348
TIME: 1:40.50 CPU: 18% major: 3 minor: 205 invol: 6892 vol: 34942

TIME: 1:40.65 CPU: 18% major: 3 minor: 206 invol: 6307 vol: 34286
TIME: 1:40.11 CPU: 18% major: 3 minor: 205 invol: 6897 vol: 35085

TIME: 1:40.45 CPU: 18% major: 3 minor: 206 invol: 6486 vol: 34424
TIME: 1:40.48 CPU: 18% major: 3 minor: 207 invol: 6897 vol: 34889

TIME: 1:40.34 CPU: 18% major: 3 minor: 204 invol: 6512 vol: 34373
TIME: 1:40.44 CPU: 18% major: 3 minor: 205 invol: 6979 vol: 35393

10001

---

time ctls -l c_10000.ct:

TIME: 0:00.09 CPU: 20% major: 0 minor: 230 invol: 3 vol: 26
TIME: 0:00.03 CPU: 53% major: 0 minor: 232 invol: 2 vol: 5

TIME: 0:00.02 CPU: 114% major: 0 minor: 231 invol: 3 vol: 6
TIME: 0:00.02 CPU: 100% major: 0 minor: 232 invol: 5 vol: 2

TIME: 0:00.01 CPU: 105% major: 0 minor: 231 invol: 3 vol: 4
TIME: 0:00.01 CPU: 84% major: 0 minor: 231 invol: 4 vol: 4

TIME: 0:00.01 CPU: 105% major: 0 minor: 232 invol: 1 vol: 6
TIME: 0:00.01 CPU: 84% major: 0 minor: 232 invol: 3 vol: 4

TIME: 0:00.01 CPU: 105% major: 0 minor: 230 invol: 4 vol: 4
TIME: 0:00.02 CPU: 100% major: 0 minor: 230 invol: 1 vol: 6

10000

/dev/md2 on /raid/heinrich type reiserfs (rw,noexec,nosuid,nodev)
100 (256*4 KB) Dateien on reiserfs

---

time create_testfiles2.sh V_100:
TIME: 0:02.15 CPU: 66% major: 7 minor: 22312 invol: 213 vol: 234

```
time ctmk c_100.ct:
TIME: 0:03.33  CPU: 29%  major: 0  minor: 135  invol: 56  vol: 572

----------

avtest V_100:

TIME: 0:02.47  CPU: 7%  major: 0  minor: 132  invol: 3  vol: 600
TIME: 0:00.13  CPU: 100%  major: 0  minor: 131  invol: 3  vol: 2

TIME: 0:02.59  CPU: 7%  major: 0  minor: 131  invol: 5  vol: 592
TIME: 0:00.13  CPU: 100%  major: 0  minor: 131  invol: 2  vol: 5

TIME: 0:02.53  CPU: 7%  major: 0  minor: 131  invol: 6  vol: 582
TIME: 0:00.14  CPU: 100%  major: 0  minor: 131  invol: 1  vol: 3

TIME: 0:02.51  CPU: 6%  major: 0  minor: 130  invol: 4  vol: 587
TIME: 0:00.13  CPU: 100%  major: 0  minor: 132  invol: 1  vol: 2

TIME: 0:02.42  CPU: 9%  major: 0  minor: 130  invol: 1  vol: 590
TIME: 0:00.14  CPU: 100%  major: 0  minor: 131  invol: 3  vol: 2

100
----------

actest c_100.ct:

TIME: 0:01.19  CPU: 18%  major: 1  minor: 134  invol: 9  vol: 300
TIME: 0:00.18  CPU: 100%  major: 0  minor: 136  invol: 2  vol: 2

TIME: 0:01.15  CPU: 21%  major: 0  minor: 136  invol: 9  vol: 319
TIME: 0:00.18  CPU: 101%  major: 0  minor: 134  invol: 7  vol: 2

TIME: 0:00.95  CPU: 21%  major: 0  minor: 134  invol: 8  vol: 380
TIME: 0:00.18  CPU: 101%  major: 0  minor: 135  invol: 1  vol: 2

TIME: 0:00.99  CPU: 23%  major: 0  minor: 135  invol: 15  vol: 368
TIME: 0:00.18  CPU: 100%  major: 0  minor: 135  invol: 1  vol: 2

TIME: 0:00.93  CPU: 23%  major: 0  minor: 135  invol: 9  vol: 391
TIME: 0:00.18  CPU: 99%  major: 0  minor: 137  invol: 2  vol: 2

100
----------

/dev/md2 on /raid/heinrich type ext3 (rw,noexec,nosuid,nodev)
1000 (256*4 KB) Dateien on ext3
----------

time create_testfiles2.sh V_1000:
TIME: 0:12.61  CPU: 79%  major: 0  minor: 215857  invol: 2131  vol: 2010

time ctmk c_1000.ct:
TIME: 0:49.04  CPU: 21%  major: 0  minor: 184  invol: 992  vol: 5820

----------

avtest V_1000:

TIME: 0:24.37  CPU: 8%  major: 0  minor: 144  invol: 52  vol: 6179
TIME: 0:24.12  CPU: 7%  major: 0  minor: 143  invol: 61  vol: 6328

TIME: 0:24.53  CPU: 7%  major: 0  minor: 143  invol: 46  vol: 6162
TIME: 0:23.97  CPU: 7%  major: 0  minor: 143  invol: 60  vol: 6363

TIME: 0:24.60  CPU: 8%  major: 0  minor: 143  invol: 49  vol: 6149
TIME: 0:24.35  CPU: 7%  major: 0  minor: 142  invol: 53  vol: 6314

TIME: 0:24.28  CPU: 7%  major: 0  minor: 143  invol: 39  vol: 6175
TIME: 0:24.07  CPU: 8%  major: 0  minor: 144  invol: 66  vol: 6350

TIME: 0:24.36  CPU: 7%  major: 0  minor: 142  invol: 54  vol: 6189
TIME: 0:24.04  CPU: 8%  major: 0  minor: 144  invol: 51  vol: 6369

1000
----------

actest c_1000.ct:

TIME: 0:09.67  CPU: 21%  major: 0  minor: 150  invol: 119  vol: 3687
TIME: 0:08.91  CPU: 23%  major: 0  minor: 150  invol: 496  vol: 4436

TIME: 0:09.65  CPU: 20%  major: 0  minor: 149  invol: 152  vol: 3488
TIME: 0:09.33  CPU: 23%  major: 0  minor: 148  invol: 361  vol: 4014

TIME: 0:09.40  CPU: 22%  major: 0  minor: 148  invol: 130  vol: 3843
TIME: 0:08.95  CPU: 26%  major: 0  minor: 149  invol: 476  vol: 4349

TIME: 0:09.91  CPU: 21%  major: 0  minor: 150  invol: 131  vol: 3316
TIME: 0:08.69  CPU: 25%  major: 0  minor: 149  invol: 472  vol: 4947

TIME: 0:09.78  CPU: 20%  major: 0  minor: 150  invol: 131  vol: 3342
TIME: 0:09.19  CPU: 24%  major: 0  minor: 148  invol: 463  vol: 4241

1000
----------

/dev/md2 on /raid/heinrich type ext3 (rw,noexec,nosuid,nodev)
10000 (256*4 KB) Dateien on ext3
----------

time create_testfiles2.sh V_10000:
TIME: 2:25.64  CPU: 71%  major: 7  minor: 2159386  invol: 27209  vol: 21794

time ctmk c_10000.ct:
TIME: 6:01.02  CPU: 32%  major: 0  minor: 498  invol: 24881  vol: 55937

----------
```

```
time ctmk c_100.ct:
TIME: 0:01.93  CPU: 62%  major: 0  minor: 134  invol: 133  vol: 453

----------

avtest V_100:

TIME: 0:01.24  CPU: 17%  major: 0  minor: 131  invol: 35  vol: 567
TIME: 0:00.13  CPU: 100%  major: 0  minor: 132  invol: 0  vol: 2

TIME: 0:01.19  CPU: 22%  major: 0  minor: 132  invol: 31  vol: 567
TIME: 0:00.13  CPU: 100%  major: 0  minor: 131  invol: 2  vol: 2

TIME: 0:01.17  CPU: 20%  major: 0  minor: 133  invol: 20  vol: 548
TIME: 0:00.13  CPU: 100%  major: 0  minor: 130  invol: 2  vol: 2

TIME: 0:01.17  CPU: 21%  major: 0  minor: 131  invol: 23  vol: 556
TIME: 0:00.13  CPU: 100%  major: 0  minor: 132  invol: 2  vol: 2

TIME: 0:01.15  CPU: 23%  major: 0  minor: 132  invol: 33  vol: 557
TIME: 0:00.13  CPU: 98%  major: 0  minor: 131  invol: 2  vol: 3

100
----------

actest c_100.ct:

TIME: 0:01.08  CPU: 20%  major: 1  minor: 134  invol: 29  vol: 352
TIME: 0:00.17  CPU: 100%  major: 0  minor: 136  invol: 1  vol: 2

TIME: 0:01.04  CPU: 17%  major: 0  minor: 135  invol: 29  vol: 339
TIME: 0:00.17  CPU: 100%  major: 0  minor: 136  invol: 2  vol: 2

TIME: 0:01.01  CPU: 20%  major: 0  minor: 135  invol: 26  vol: 355
TIME: 0:00.17  CPU: 100%  major: 0  minor: 136  invol: 1  vol: 2

TIME: 0:01.01  CPU: 21%  major: 0  minor: 135  invol: 34  vol: 379
TIME: 0:00.17  CPU: 100%  major: 0  minor: 136  invol: 2  vol: 2

TIME: 0:01.04  CPU: 21%  major: 0  minor: 134  invol: 36  vol: 331
TIME: 0:00.17  CPU: 100%  major: 0  minor: 136  invol: 2  vol: 2

100
----------

/dev/md2 on /raid/heinrich type reiserfs (rw,noexec,nosuid,nodev)
1000 (256*4 KB) Dateien on reiserfs
----------

time create_testfiles2.sh V_1000:
TIME: 0:12.51  CPU: 95%  major: 0  minor: 214602  invol: 2153  vol: 2005

time ctmk c_1000.ct:
TIME: 0:32.86  CPU: 43%  major: 0  minor: 184  invol: 2876  vol: 5013

----------

avtest V_1000:

TIME: 0:11.57  CPU: 22%  major: 0  minor: 143  invol: 345  vol: 6078
TIME: 0:11.18  CPU: 21%  major: 0  minor: 143  invol: 647  vol: 5907

TIME: 0:11.76  CPU: 22%  major: 0  minor: 144  invol: 357  vol: 6093
TIME: 0:11.51  CPU: 21%  major: 0  minor: 144  invol: 636  vol: 5963

TIME: 0:11.53  CPU: 23%  major: 0  minor: 143  invol: 365  vol: 6040
TIME: 0:11.17  CPU: 21%  major: 0  minor: 142  invol: 661  vol: 5953

TIME: 0:11.67  CPU: 21%  major: 0  minor: 144  invol: 299  vol: 6015
TIME: 0:11.11  CPU: 22%  major: 0  minor: 144  invol: 651  vol: 5971

TIME: 0:11.64  CPU: 22%  major: 0  minor: 143  invol: 359  vol: 6064
TIME: 0:11.18  CPU: 21%  major: 0  minor: 142  invol: 650  vol: 5972

1000
----------

actest c_1000.ct:

TIME: 0:10.55  CPU: 21%  major: 0  minor: 148  invol: 366  vol: 3277
TIME: 0:09.08  CPU: 30%  major: 0  minor: 149  invol: 620  vol: 4479

TIME: 0:10.37  CPU: 22%  major: 0  minor: 149  invol: 363  vol: 3195
TIME: 0:09.05  CPU: 30%  major: 0  minor: 148  invol: 647  vol: 4472

TIME: 0:10.33  CPU: 22%  major: 0  minor: 149  invol: 376  vol: 3198
TIME: 0:08.87  CPU: 32%  major: 0  minor: 150  invol: 601  vol: 4492

TIME: 0:10.30  CPU: 22%  major: 0  minor: 149  invol: 380  vol: 3246
TIME: 0:08.88  CPU: 31%  major: 0  minor: 149  invol: 660  vol: 4643

TIME: 0:10.24  CPU: 21%  major: 0  minor: 148  invol: 339  vol: 3306
TIME: 0:08.88  CPU: 33%  major: 0  minor: 150  invol: 577  vol: 4664

1000
----------

/dev/md2 on /raid/heinrich type reiserfs (rw,noexec,nosuid,nodev)
10000 (256*4 KB) Dateien on reiserfs
----------

time create_testfiles2.sh V_10000:
TIME: 2:13.94  CPU: 91%  major: 7  minor: 2139510  invol: 25256  vol: 20050

time ctmk c_10000.ct:
TIME: 5:54.49  CPU: 41%  major: 0  minor: 470  invol: 36816  vol: 48184

----------
```

avtest V_10000:

TIME: 2:08.85	CPU: 15%	major: 0	minor: 287	invol: 5264	vol: 67498
TIME: 2:07.23	CPU: 16%	major: 0	minor: 286	invol: 5216	vol: 68056
TIME: 2:07.81	CPU: 15%	major: 0	minor: 286	invol: 5146	vol: 67976
TIME: 2:06.27	CPU: 16%	major: 0	minor: 286	invol: 5244	vol: 68106
TIME: 2:07.81	CPU: 15%	major: 0	minor: 287	invol: 5146	vol: 67780
TIME: 2:07.55	CPU: 16%	major: 0	minor: 286	invol: 5327	vol: 67944
TIME: 2:07.69	CPU: 15%	major: 0	minor: 285	invol: 5055	vol: 67719
TIME: 2:07.41	CPU: 16%	major: 0	minor: 286	invol: 6004	vol: 67644
TIME: 2:07.69	CPU: 15%	major: 0	minor: 286	invol: 5131	vol: 67618
TIME: 2:07.34	CPU: 16%	major: 0	minor: 286	invol: 5965	vol: 67740

10000

---

actest c_10000.ct:

TIME: 1:38.63	CPU: 19%	major: 0	minor: 294	invol: 5739	vol: 34656
TIME: 1:38.55	CPU: 19%	major: 1	minor: 293	invol: 6094	vol: 34900
TIME: 1:39.62	CPU: 19%	major: 1	minor: 293	invol: 5996	vol: 33832
TIME: 1:38.69	CPU: 19%	major: 1	minor: 294	invol: 6142	vol: 34886
TIME: 1:39.09	CPU: 19%	major: 1	minor: 294	invol: 5753	vol: 34566
TIME: 1:38.76	CPU: 19%	major: 1	minor: 293	invol: 6163	vol: 35284
TIME: 1:39.02	CPU: 19%	major: 1	minor: 293	invol: 5836	vol: 34157
TIME: 1:38.00	CPU: 19%	major: 1	minor: 294	invol: 6092	vol: 35181
TIME: 1:39.03	CPU: 19%	major: 1	minor: 293	invol: 5831	vol: 34924
TIME: 1:39.09	CPU: 19%	major: 1	minor: 293	invol: 6234	vol: 34725

10000

---

avtest V_10000:

TIME: 1:59.18	CPU: 21%	major: 0	minor: 287	invol: 6715	vol: 61250
TIME: 1:55.97	CPU: 21%	major: 0	minor: 285	invol: 7024	vol: 60459
TIME: 2:00.25	CPU: 22%	major: 0	minor: 286	invol: 6739	vol: 61280
TIME: 1:55.36	CPU: 21%	major: 0	minor: 285	invol: 7309	vol: 60491
TIME: 1:58.85	CPU: 22%	major: 0	minor: 286	invol: 6832	vol: 61173
TIME: 1:55.14	CPU: 21%	major: 0	minor: 286	invol: 6859	vol: 60354
TIME: 1:59.83	CPU: 21%	major: 0	minor: 286	invol: 6821	vol: 61440
TIME: 1:55.39	CPU: 21%	major: 0	minor: 286	invol: 6984	vol: 60881
TIME: 1:59.10	CPU: 21%	major: 0	minor: 286	invol: 6581	vol: 61391
TIME: 1:55.54	CPU: 22%	major: 0	minor: 287	invol: 7316	vol: 60902

10000

---

actest c_10000.ct:

TIME: 1:45.57	CPU: 24%	major: 0	minor: 293	invol: 7122	vol: 32630
TIME: 1:45.93	CPU: 25%	major: 1	minor: 293	invol: 7478	vol: 33173
TIME: 1:46.14	CPU: 24%	major: 1	minor: 293	invol: 7219	vol: 32487
TIME: 1:46.19	CPU: 25%	major: 1	minor: 292	invol: 7465	vol: 33523
TIME: 1:46.00	CPU: 24%	major: 1	minor: 293	invol: 7002	vol: 32976
TIME: 1:46.13	CPU: 25%	major: 1	minor: 294	invol: 7374	vol: 33258
TIME: 1:45.81	CPU: 24%	major: 1	minor: 293	invol: 7111	vol: 32862
TIME: 1:46.24	CPU: 25%	major: 1	minor: 292	invol: 7396	vol: 33421
TIME: 1:45.21	CPU: 24%	major: 1	minor: 294	invol: 6859	vol: 33241
TIME: 1:44.84	CPU: 24%	major: 1	minor: 292	invol: 7256	vol: 33948

10000

Testergebnistabellen

EXT3 Erstellung cold 4 KByte

	Verzeichnis	Container	TAR
10	60	0	20
100	220	20	210
1000	1940	220	3570
10000	19190	2760	58100
100000	195090	41470	1016290
1000000	2108760	665070	12566000

ReiserFS Erstellung cold 4 KByte

	Verzeichnis	Container	TAR
10	360	0	80
100	220	20	50
1000	2010	230	280
10000	19890	2310	2770
100000	200500	28960	28970
1000000	2017360	361180	333820

EXT3 Erstellung cold 1024 KByte

	Verzeichnis	Container	TAR
10	590	320	360
100	1320	3440	2980
1000	12310	47100	50060
10000	146100	356050	509170

ReiserFS Erstellung cold 1024 KByte

	Verzeichnis	Container	TAR
10	770	160	200
100	2150	1880	2470
1000	12690	31870	29690
10000	134640	354710	319620

EXT3 Auflisten cold ohne Meta 4 KByte

	Verzeichnis	Container	TAR
10	0	0	0
100	10	6	20
1000	4	2	60
10000	108	6	500
100000	1224	52	4536
1000000	50568	382	51544

ReiserFS Auflisten cold ohne Meta 4 KByte

	Verzeichnis	Container	TAR
10	0	0	0
100	0	0	0
1000	4	2	48
10000	60	14	456
100000	576	54	4420
1000000	6486	362	43744

EXT3 Auflisten hot ohne Meta 4 KByte

	Verzeichnis	Container	TAR
10	0	0	0
100	0	0	0
1000	0	0	10
10000	80	0	120
100000	1064	20	1244
1000000	13464	198	49728

ReiserFS Auflisten hot ohne Meta 4 KByte

	Verzeichnis	Container	TAR
10	0	0	0
100	0	0	0
1000	0	0	10
10000	40	0	120
100000	512	20	1238
1000000	6034	194	43934

EXT3 Auflisten cold mit Meta 4 KByte

	Verzeichnis	Container	TAR
10	0	0	0
100	8	0	28
1000	78	6	54
10000	654	20	484
100000	23100	182	4512
1000000	432104	1856	47990

ReiserFS Auflisten cold mit Meta 4 KByte

	Verzeichnis	Container	TAR
10	0	0	0
100	0	0	8
1000	52	0	64
10000	624	20	488
100000	6460	192	4534
1000000	89394	1878	45348

EXT3 Auflisten hot mit Meta 4 KByte

	Verzeichnis	Container	TAR
10	0	0	0
100	0	0	0
1000	40	0	20
10000	410	14	200
100000	4368	174	1998
1000000	157878	1740	49540

ReiserFS Auflisten hot mit Meta 4 KByte

	Verzeichnis	Container	TAR
10	0	0	0
100	0	0	0
1000	32	0	20
10000	374	12	200
100000	3880	176	1998
1000000	45234	1756	45528

EXT3 Auflisten cold ohne Meta 1024 KByte

	Verzeichnis	Container	TAR
10	2	10	114
100	2	6	1018
1000	22	6	9718
10000	704	30	97510

ReiserFS Auflisten cold ohne Meta 1024 KByte

	Verzeichnis	Container	TAR
10	2	2	112
100	0	2	1028
1000	8	16	10352
10000	92	22	100562

EXT3 Auflisten hot ohne Meta 1024 KByte

	Verzeichnis	Container	TAR
10	0	0	10
100	0	0	92
1000	0	0	9012
10000	80	0	97628

ReiserFS Auflisten hot ohne Meta 1024 KByte

	Verzeichnis	Container	TAR
10	0	0	10
100	0	0	90
1000	0	0	9058
10000	40	0	100120

EXT3 Auflistenc cold mit Meta 1024 KByte

	Verzeichnis	Container	TAR
10	18	4	124
100	4	4	1004
1000	174	6	9742
10000	1586	32	97428

ReiserFS Auflisten cold mit Meta 1024 KByte

	Verzeichnis	Container	TAR
10	0	0	120
100	20	8	1036
1000	1426	12	10222
10000	16632	28	100402

EXT3 Auflisten hot mit Meta 1024 KByte

	Verzeichnis	Container	TAR
10	0	0	10
100	0	0	90
1000	40	0	8974
10000	414	18	97866

ReiserFS Auflisten hot mit Meta 1024 KByte

	Verzeichnis	Container	TAR
10	0	0	1
100	0	0	90
1000	40	0	9080
10000	388	18	100292

EXT3 Lesen cold 4 KByte

	Verzeichnis	Container
100	136	18
1000	692	56
10000	3146	424
100000	39204	3882
1000000	536702	38700

ReiserFS Lesen cold 4 KByte

	Verzeichnis	Container
100	56	10
1000	196	42
10000	1788	424
100000	18490	3982
1000000	208218	40060

EXT3 Lesen hot 4 KByte

	Verzeichnis	Container
100	0	0
1000	30	10
10000	330	150
100000	3406	1606
1000000	462582	38818

ReiserFS Lesen hot 4 KByte

	Verzeichnis	Container
100	0	0
1000	28	10
10000	354	148
100000	3774	1542
1000000	212180	40382

EXT3 Lesen cold 1204 KByte

	Verzeichnis	Container
100	2506	1014
1000	24416	9688
10000	127904	96748

ReiserFS Lesen cold 1024 KByte

	Verzeichnis	Container
100	1172	1076
1000	11550	10306
10000	119930	106040

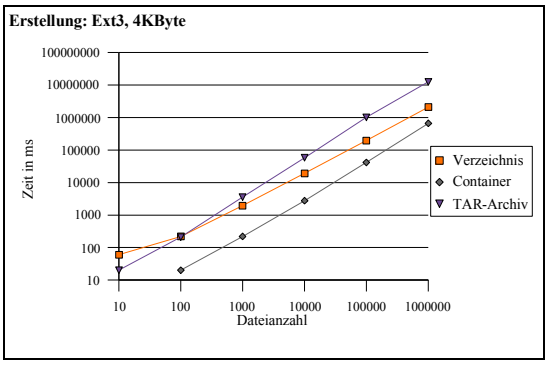
EXT3 Lesen hot 1204 KByte

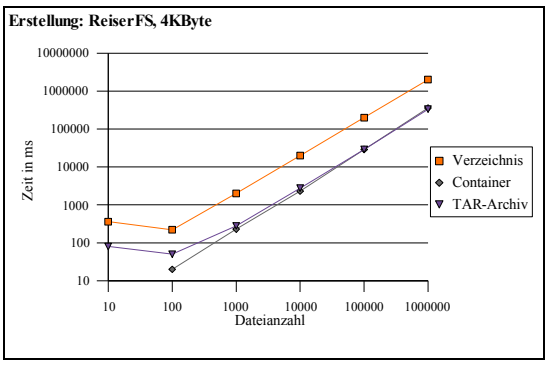
	Verzeichnis	Container
100	132	180
1000	24218	9110
10000	127338	98658

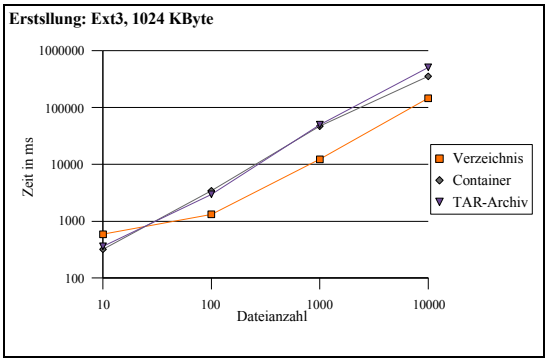
ReiserFS Lesen hot 1024 KByte

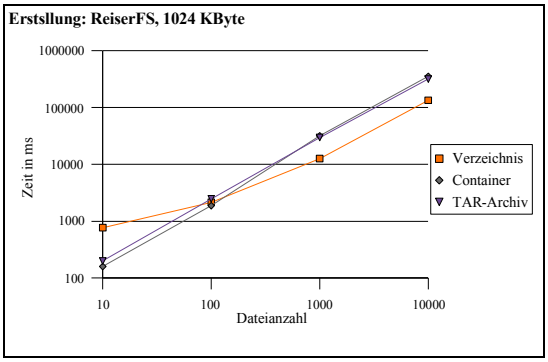
	Verzeichnis	Container
100	132	170
1000	11190	9010
10000	115390	106234

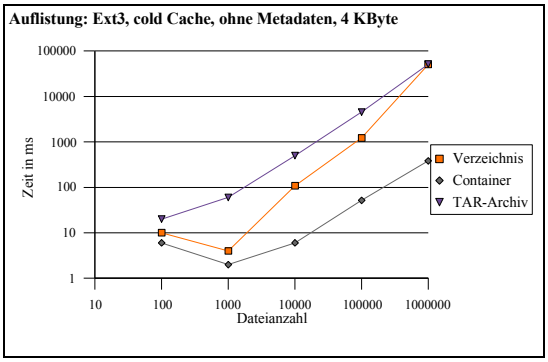
Testergebnisdiagramme

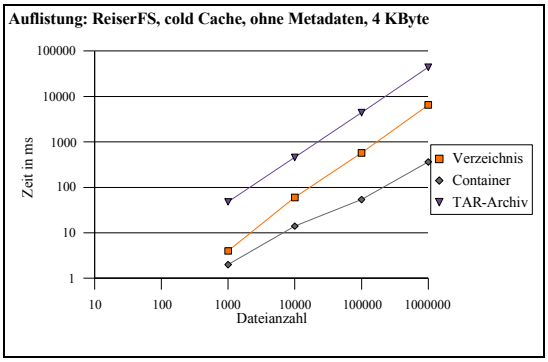


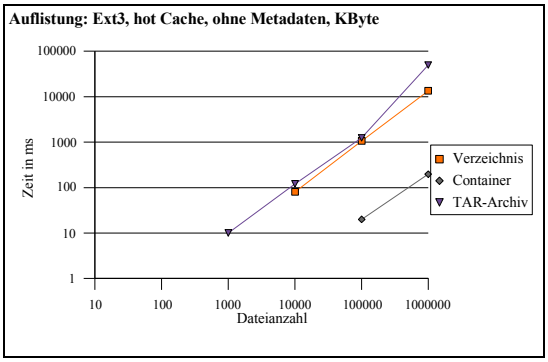


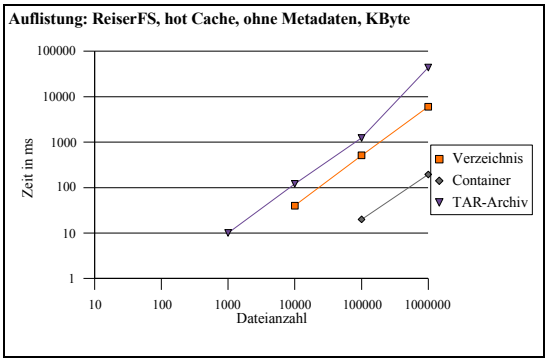


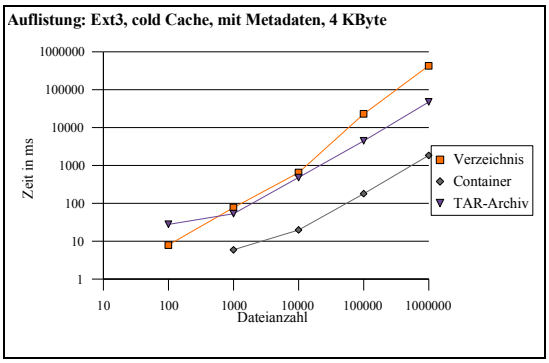


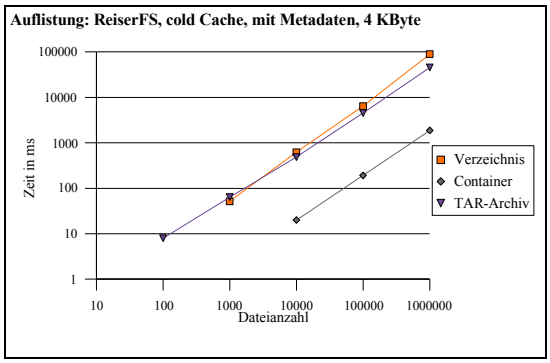


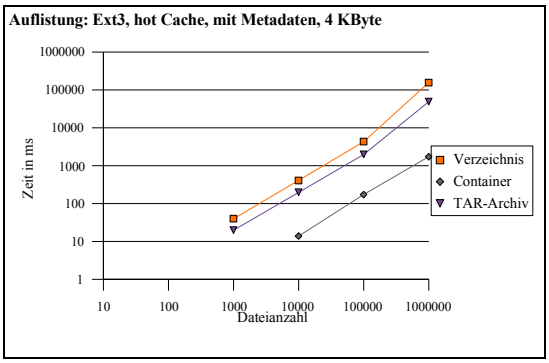


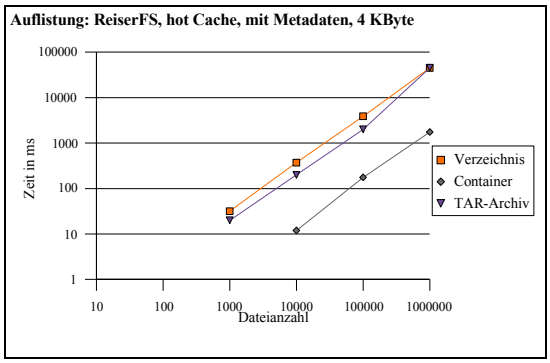


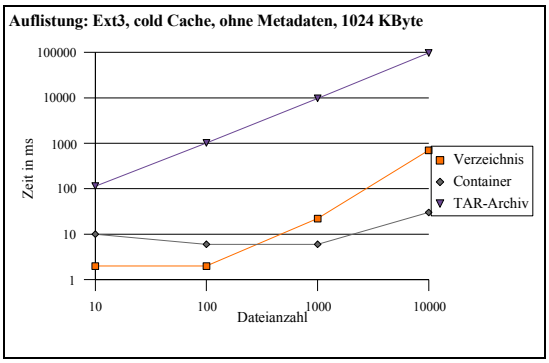


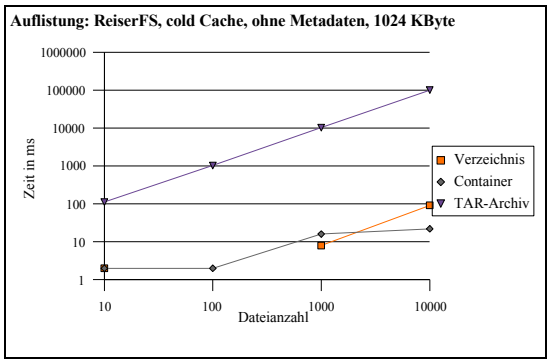

Auflistung: Ext3, cold Cache, mit Metadaten, 4 KByte
Zeit in ms
1000000
100000
10000
1000
100
10
1
10
100
1000
10000
100000
1000000
Dateianzahl
Verzeichnis
Container
TAR-Archiv

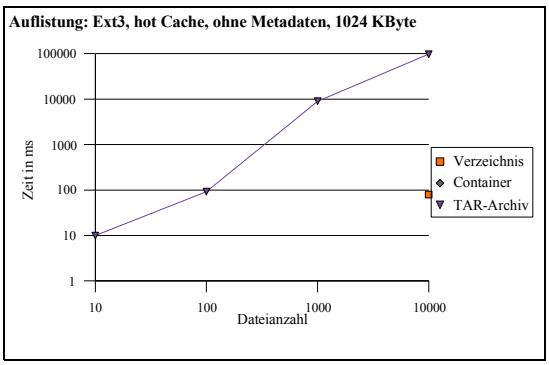

Auflistung: ReiserFS, cold Cache, mit Metadaten, 4 KByte
Zeit in ms
100000
10000
1000
100
10
1
10
100
1000
10000
100000
1000000
Dateianzahl
Verzeichnis
Container
TAR-Archiv

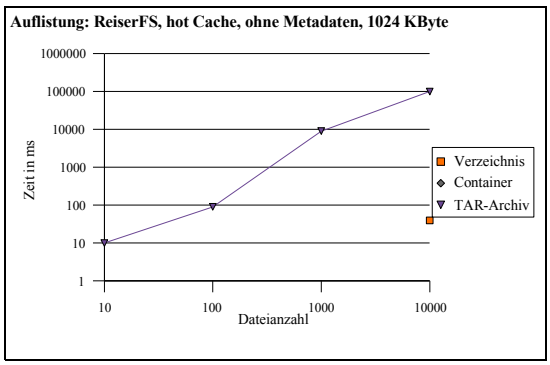

Auflistung: Ext3, hot Cache, mit Metadaten, 4 KByte
Zeit in ms
1000000
100000
10000
1000
100
10
1
10
100
1000
10000
100000
1000000
Dateianzahl
Verzeichnis
Container
TAR-Archiv


Auflistung: ReiserFS, hot Cache, mit Metadaten, 4 KByte
Zeit in ms
100000
10000
1000
100
10
1
10
100
1000
10000
100000
1000000
Dateianzahl
Verzeichnis
Container
TAR-Archiv

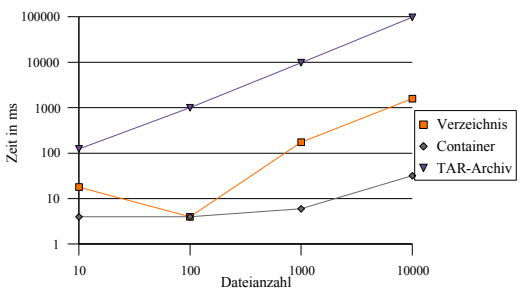

Auflistung: Ext3, cold Cache, ohne Metadaten, 1024 KByte
Zeit in ms
100000
10000
1000
100
10
1
10
100
1000
10000
Dateianzahl
Verzeichnis
Container
TAR-Archiv


Auflistung: ReiserFS, cold Cache, ohne Metadaten, 1024 KByte
Zeit in ms
1000000
100000
10000
1000
100
10
1
10
100
1000
10000
Dateianzahl
Verzeichnis
Container
TAR-Archiv

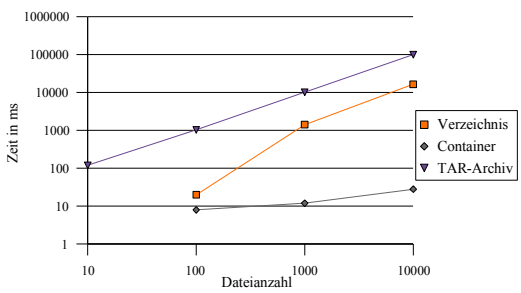

Auflistung: Ext3, hot Cache, ohne Metadaten, 1024 KByte
Zeit in ms
100000
10000
1000
100
10
1
10
100
1000
10000
Dateianzahl
Verzeichnis
Container
TAR-Archiv


Auflistung: ReiserFS, hot Cache, ohne Metadaten, 1024 KByte
Zeit in ms
1000000
100000
10000
1000
100
10
1
10
100
1000
10000
Dateianzahl
Verzeichnis
Container
TAR-Archiv

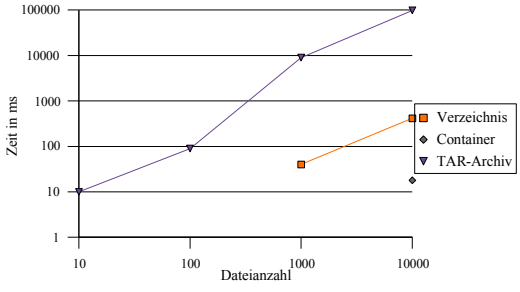
Auflistung: Ext3, cold Cache, mit Metadaten, 1024 KByte



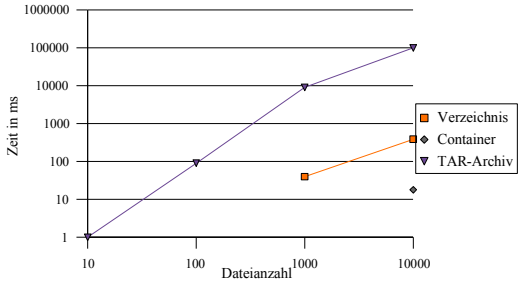
Auflistung: ReiserFS, cold Cache, mit Metadaten, 1024 KByte



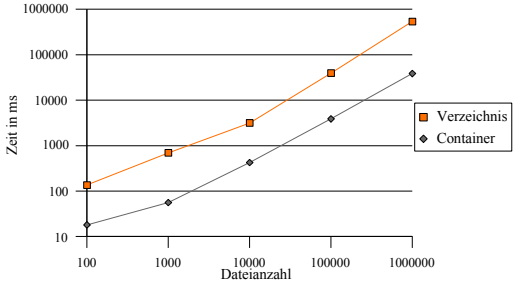
Auflistung: Ext3, hot Cache, mit Metadaten, 1024 KByte



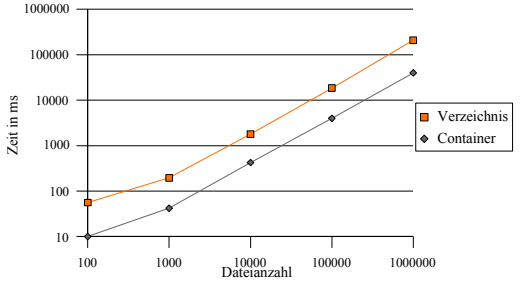
Auflistung: ReiserFS, hot Cache, mit Metadaten, 1024 KByte



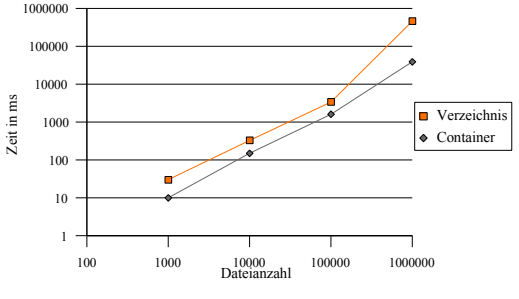
Lesen: Ext3, cold Cache, 4 KByte



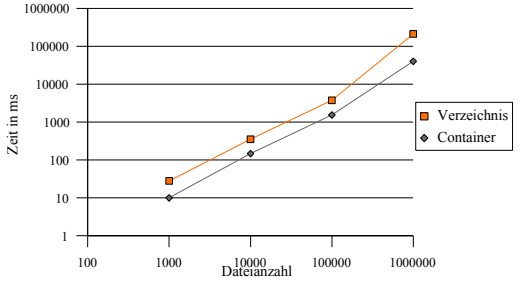
Lesen: ReiserFS, cold Cache, 4 KByte

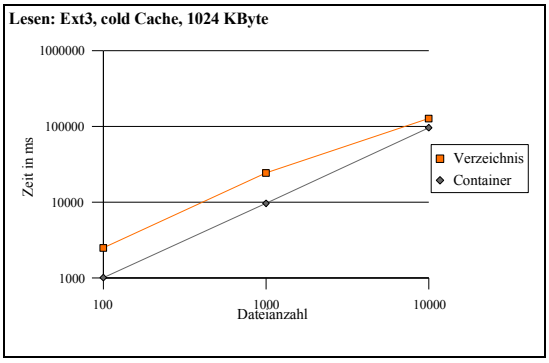


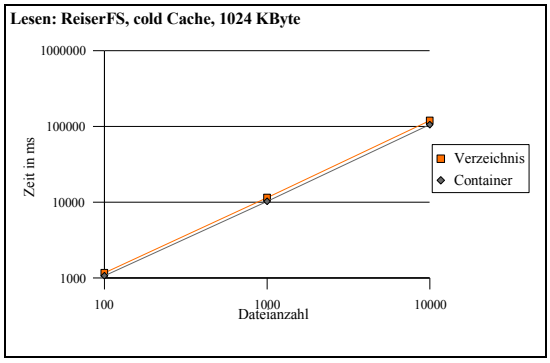
Lesen: Ext3, hot Cache, 4 KByte

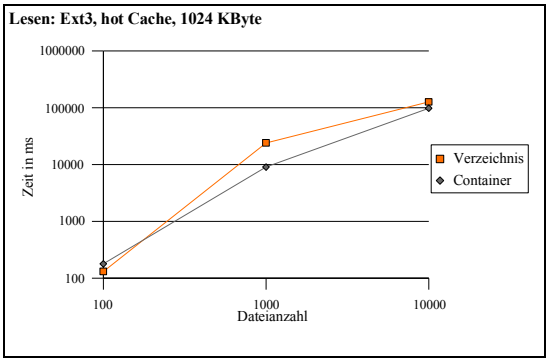


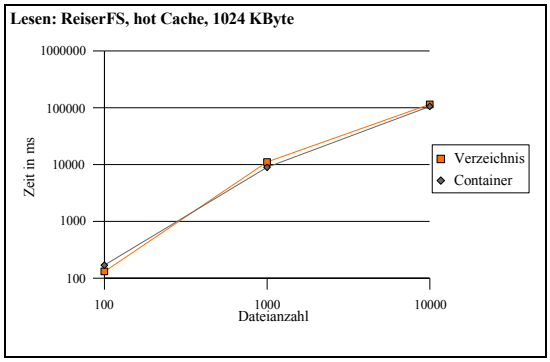
Lesen: ReiserFS, hot Cache, 4 KByte




Lesen: Ext3, cold Cache, 1024 KByte
1000000
100000
10000
1000
Zeit in ms
Verzeichnis
Container
100
1000
10000
Dateianzahl


Lesen: ReiserFS, cold Cache, 1024 KByte
1000000
100000
10000
1000
Zeit in ms
Verzeichnis
Container
100
1000
10000
Dateianzahl


Lesen: Ext3, hot Cache, 1024 KByte
1000000
100000
10000
1000
100
Zeit in ms
Verzeichnis
Container
100
1000
10000
Dateianzahl


Lesen: ReiserFS, hot Cache, 1024 KByte
1000000
100000
10000
1000
100
Zeit in ms
Verzeichnis
Container
100
1000
10000
Dateianzahl

Quelltext *libct.h*

```
/*
##############################################################################################
#
# Name: libct.h
#
# Description: API definition for container file usage
#
# Author: Hendrik Heinrich  hendrikheinrich@gmx.de
#
# created: 27.07.2007
#
# last modified: 30.09.2007
#
##############################################################################################
*/

#ifndef _LIBCT_H
#define _LIBCT_H
#include <sys/types.h>
#include <fcntl.h>

struct containerDescriptor;
struct cFile;
struct indexArray;

/* ct_open
 *   opens a container file for reading access and returns a structure
 *   containerDescriptor representing it.
 * expects the name of a container file
 * returns a structure containerDescriptor
 */
struct containerDescriptor* ct_open(char* containerName);

/* ct_close
 *   closes a container file and frees allocated memory for the metadata
 *expects a structure containerDescriptor and
 *returns "0" in case of success and "-1" for failure
 */
int ct_close(struct containerDescriptor* containerDescriptor);

/*ct_readdir
 *   reads the following file name of an contained file
 *   ATTENTION! prevName must point to an address in fNameArray
 * expects a structure containerDescriptor
 * expects the file name of the previous file or NULL for demanding
 *        name of first contained file
 * returns the name of the following file
 */
char* ct_readdir(struct containerDescriptor* containerDescriptor, char* prevName);
```

```
/* ct_file_locate
 *   locates the corresponding index block to a filename.
 * expects a NULL-pointer to an indexArray structur. This pointer will hold
 *         the address of the located file in case of existence and
 * expects a containerDescriptor and
 * returns "0" for successful searching and "-1" for failure
 */
struct indexArray* ct_file_locate(struct containerDescriptor* containerDescriptor, char* fName);

/* ct_file_open
 *   opens a contained file by returning a structur cFile as representative
 * expects a container file descriptor and
 * expects a structure indexArray providing file metadata
 * returns a structur cFile
 */
struct cFile* ct_file_open( struct containerDescriptor* containerDescriptorm
                            struct indexArray* locatedFile);

/*ct_file_size
 *   returns size of given contained file
 *expects structure cFile as file representation
 *returns size in byte
 */
ulong32 ct_file_size(struct cFile* cFile);

/* ct_file_read
 *   reads n bytes of file content to buffer
 * expects cFile as reprentation of file to be read
 * expects buffer
 * expects nbytes number of bytes to be read
 * returns number of bytes read
 */
size_t ct_file_read(struct cFile* cFile, void* buffer, size_t nbyte);

/* ct_file_close
 *   closes open contained file by freeing memory for structure cFile
 * expects structure cFile
 */
void ct_file_close(struct cFile* cFile);

/* ct_file_lseek
 *   moves read/write pointer toposition determined by offset
 * expects structure cFile
 * expects offset to set to
 * expects whence
 * returns new position of read/write pointer in byte
 */
off_t ct_file_lseek(struct cFile* cFile, off_t offset, int whence);

#endif
```

Quelltext *libct-internal.h*

```
/*
##########################################################################################
#
# Name: libct-internal.h
#
# Description: structure and internal procedure definitions for container API usage
#
# Author: Hendrik Heinrich  hendrikheinrich@gmx.de
#
# created: 27.07.2007
#
# last modified: 30.09.2007
#
##########################################################################################
*/

#ifndef _LIBCT_INTERNAL_H
#define _LIBCT_INTERNAL_H

#include <sys/types.h>
#include <fcntl.h>


typedef unsigned long int ulong32;

/*  structure to hold container file header information */
struct header{
  ulong32 numFiles;
  ulong32 sumNameLength;
  off_t indexOffset;};

/* struture as array element to hold file information in a container file */
struct indexArray{
  ulong32 fNameOffset;
  ulong32 fSize;
  off_t fOffset;};

/* structure to be used to represent an open container file */
struct containerDescriptor{
  int cFD;
  struct header* header;
  char* fNameArray;
  struct indexArray* indexArray;};

/* structure to be used to represent an open contained file */
struct cFile{
  struct containerDescriptor* containerDescriptor;
  struct indexArray* fData;
  off_t curPosition;};
```

```
/* create_containerHeader
 *   allocates space for a header structure for the purose of
 *   containing information about how many bytes do all file names
 *   need and how many files are to be contained and
 * returns a pointer to this structure
 */
struct header* create_containerHeader(void);

/* read_containerHeader
 *   reads the Header of a container file and writes its content into a
 *   header structure.
 * expects the file name of the container file to be read and
 * expects a header structer to write into
 * retruns "0" for successful reading and "-1" for failure
 */
int read_containerHeader(int cFD, struct header* readHeader);

/* read_containerIndex
 *   reads the index from a container file and writes the content into a
 *   file name array containing all names of contained files and remaining file
 *   informations for all contained files into an array of structures.
 *   The a file name offset is use to determine the location of the
 *   corresponding file name for each index entry.
 * expects a file descriptor for the file to read from and
 * expects an index array to write to and
 * expects an file name array to write to and
 * expects a header structure and
 * returns "0" for successful reading and "-1" for failure
 */
int read_containerIndex( int cFD, struct indexArray* indexArray, char* fNameArray,
                         struct header* header);

/* read_containedFileNames
 *   reads the file names from container file into a file name array.
 * expects a file descriptor for the container file to read from
 * expects a file name array to write to
 * exptects the header of the container file
 * returns "0" for successful reading an "-1" for failure
 */
int read_containedFileNames(int cFD, char* fNameArray, struct header* header);

/* read_containedFileIndex
 *   reads the file information or contained files from the container
 *   file into an index array
 * expects a file descriptor for the container file
 * expects an index array to write to
 * expects the header of the container file
 * returns "0" for successful reading and "-1" for failure
 */
int read_containedFileIndex(int cFD, struct indexArray* indexArray, struct header* header);

#endif
```

Quelltext *ctest.sh*

```
#!/bin/bash

if [ -z "$1" ] || [ -z "$2" ]
then
    echo "Usage: $(basename "$0") <start number of files> <end number of files>"
    exit
fi

START="$1"
STOP="$2"
P="/home/hheinric/Programme_Bachelorarbeit/bin"
WP="/home/hheinric"
MP="/raid/heinrich"
D="/dev/md2"
T="time"

export TIME="TIME: %E   CPU: %P  \tmajor: %F\tminor: %R\tinvol: %c\tvol: %w"

for((k=1; k<=2; k++))
do
    if(($k == 1))
    then
        l=4
        COUNT=$STOP
    fi
    if(($k == 2))
    then
        l=1024
        if(( $STOP > 10000))
        then
            COUNT=10000
        fi
    fi

    # "1" -> test on ext3, "2" -> test on reiserfs
    for((j=1; j<=2; j++))
    do
        for((i=$START; i<=$COUNT; i=i*10))
        do
            umount $MP

            if(($j == 1))
            then
                mkfs.ext3 -O dir_index $D &> /dev/null
                FS="on ext3"
                mount -t ext3 -o defaults,noauto,users $D $MP
            fi

            if(($j == 2))
```

```
    then
        mkfs.reiserfs -q $D &> /dev/null
        FS="on reiserfs"
        mount -t reiserfs -o defaults,noauto,users $D $MP
    fi

    mount | grep $D

echo "$i ($l KB) Dateien $FS"
echo ----------------------------------------------------------------------------------------------
echo ""

echo "time create_testfiles.sh V_$i:"
$T $P/create_testfiles.sh $WP/test/V_$i $i $l
echo ""

umount $MP
mount $MP

echo "time ctmk C_$i.ct:"
$T $P/ctmk $WP/test/V_$i $WP/test/C_$i > /dev/null
echo ""

umount $MP
mount $MP

echo "time tar -cvf V_$i:"
$T tar -cvf $WP/test/T_$i.tar $WP/test/V_$i > /dev/null

echo ----------------------------------------------------------------------------------------------

umount $MP
mount $MP

echo ""
echo "time ls -1 V_$i:"
echo ""

for((a=1;a<=5;a++))
do
    $T ls -1 $WP/test/V_$i > /dev/null

    $T ls -1 $WP/test/V_$i > /dev/null

    echo ""

    if(($a==5))
    then
        ls -1 $WP/test/V_$i | wc -l
    fi

    umount $MP
    mount $MP
done
```

```
echo ----------------------------------------------------------------------------------------
echo ""

echo "time tar -tf T_$i.tar:"
echo ""

for((a=1; a<=5;a++))
do
    $T tar -tf $WP/test/T_$i.tar > /dev/null

    $T tar -tf $WP/test/T_$i.tar > /dev/null

    echo ""

    if(($a==5))
    then
        tar -tf $WP/test/T_$i.tar | wc -l
    fi

    umount $MP
    mount $MP
done

echo ----------------------------------------------------------------------------------------
echo ""

echo "time ctls C_$i.ct:"
echo ""

for((a=1; a<=5;a++))
do
    $T $P/ctls $WP/test/C_$i.ct >/dev/null

    $T $P/ctls $WP/test/C_$i.ct >/dev/null

    echo ""

    if(($a==5))
    then
        $P/ctls $WP/test/C_$i.ct |wc -l
    fi

    umount $MP
    mount $MP
done

echo ----------------------------------------------------------------------------------------
echo ""

echo "time ls -l V_$i:"
echo ""

for((a=1; a<=5;a++))
do
```

```
    $T ls -l $WP/test/V_$i > /dev/null

    $T ls -l $WP/test/V_$i > /dev/null

    echo ""

    if(($a==5))
    then
        ls -l $WP/test/V_$i | wc -l
    fi

    umount $MP
    mount $MP
done

echo ----------------------------------------------------------------------------------------
echo ""

echo "time tar -tvf T_$i.tar:"
echo ""

for((a=1; a<=5;a++))
do
    $T tar -tvf $WP/test/T_$i.tar > /dev/null

    $T tar -tvf $WP/test/T_$i.tar > /dev/null

    echo ""

    if(($a==5))
    then
        tar -tvf $WP/test/T_$i.tar | wc -l
    fi

    umount $MP
    mount $MP
done

echo ----------------------------------------------------------------------------------------
echo ""

echo "time ctls -l C_$i.ct:"
echo ""

for((a=1; a<=5;a++))
do
    $T $P/ctls -l $WP/test/C_$i.ct > /dev/null

    $T $P/ctls -l $WP/test/C_$i.ct > /dev/null

    echo ""

    if(($a==5))
    then
```

```
        $P/ctls -l $WP/test/C_$i.ct | wc -l
    fi

    umount $MP
    mount $MP
done

echo ----------------------------------------------------------------------------------------
    echo ""

    echo "apiVTest V_$i:"
    echo ""

    for((a=1;a<=5;a++))
    do


    $T $P/apiVTest $WP/test/V_$i/ > /dev/null
        $T $P/apiVTest $WP/test/V_$i/ > /dev/null

    echo ""

    if(($a==5))
    then
        $P/apiVTest $WP/test/V_$i/| wc -l
    fi

    umount $MP
    mount $MP
    done

    echo ----------------------------------------------------------------------------------------
    echo ""


    echo "apiCTest C_$i.ct:"
    echo ""

    for((a=1; a<=5;a++))
    do
    $T $P/apiCTest $WP/test/C_$i.ct > /dev/null

    $T $P/apiCTest $WP/test/C_$i.ct > /dev/null

    echo ""

    if(($a==5))
    then
        $P/apiCTest $WP/test/C_$i.ct | wc -l
    fi

    umount $MP
    mount $MP
    done
```

```
            echo ----------------------------------------------------------------------------------------
            echo ""
        done
    done
done
umount $MP

mkfs.ext3 -O dir_index $D &> /dev/null

mount -t ext3 -o defaults,noauto,users $D $MP
```

Quelltext *create_testfiles.sh*

```
#!/bin/bash

if [ -z "$1" ] || [ -z "$2" ] || [ -z "$3" ]
then
     echo "Usage: $(basename "$0") <directory> <file-count> <file-size>"
     exit
fi

DIRECTORY="$1"
FILECOUNT="$2"
C="$3"
i="0"
mkdir $DIRECTORY
cd "$DIRECTORY" || exit 1

dd if=/dev/urandom of=test_file_000000 bs=4096 count=$(($C/4)) 2> /dev/null

for((D5 = 0; D5 < 10; D5++))
do
 for((D4 = 0; D4 < 10; D4++))
 do
  for((D3 = 0; D3 < 10; D3++))
  do
   for((D2 = 0; D2 < 10; D2++))
   do
    for((D1 = 0; D1 < 10; D1++))
    do
     for((D0 = 0; D0 < 10; D0++))
     do
      if(($i < $FILECOUNT))
      then
       cp test_file_000000 test_file_$D6$D5$D4$D3$D2$D1$D0 2> /dev/null;
       i=$(($i+1));
      else
       exit 0
      fi
     done;
    done;
   done;
  done;
 done;
done;
```

Erklärung zur Bachelorarbeit

Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

.....

Abgabe Datum: 30. September 2007