圧縮ブロックデバイスにおけるファイルシステム最適化

北川 健司† 丹 英之† 阿部 大将† 千葉 大作† 須崎 有康‡ 飯島 賢吾‡ 八木 豊志樹‡
†{kitagake, tanh, abeda, chibad}@alpha.co.jp ‡{k.suzaki, k-iijima, yagi-toshiki}@aist.go.jp
†株式会社 アルファシステムズ ‡独立行政法人 産業技術総合研究所

概要： 近年，cloop や SquashFS などのブロック単位で圧縮を行い，膨大なコンテンツを圧縮して提供する機構が，ライブ CD を中心に広く利用されている．必要とする部分を参照するには伸張プロセスを必要とするが，ライブ CD のようにデバイス容量が制限されている場合には有効な手段と言える．しかし，ブート時に読み出しブロックが分散しているとシークが多発して起動が遅くなる．そこで我々は，ブロック圧縮された仮想デバイス上のファイルシステムを最適化するツール Ext2optimizer を開発した．これにより起動時に読み出されるブロックを集約し，シークを抑えて読み出す高速起動が可能となった．本論文では，Ext2optimizer の設計・実装とその有効性を示す．

1. はじめに

近年，cloop[1]や SquashFS[2]などのブロック単位で圧縮を行い，膨大なコンテンツを圧縮して提供する機構が，ライブ CD を中心に広く利用されている．ライブ CD の特徴は，ゲーム機のような感覚で PC が使える点である．パソコン初学者にとって敷居の高かったインストール作業が必須でなくなり，Linux に触れるまでの障壁を取り除くことができたと言える．最近では，ライブ CD で有名な KNOPPIX[3]の起動時間を半分にした Accelerated-KNOPPIX[4]が世の中に出回り，ライブ CD の利便性に拍車を掛けている．

KNOPPIX とは，ドイツの Klaus Knopper 氏により開発されている Debian GNU/Linux ベースの 1CD Linux ディストリビューションであり，産業技術総合研究所(AIST)で日本語化，及び配布[5]が行われている．

KNOPPIX の CD メディアには，ブートローダ(isolinux)，カーネル(linux)，RAM ディスク(minirt.gz)，ルートファイルシステム(cloop イメージファイル)が格納されており，起動時に cloop ファイルをルートファイルシステムとしてループバックマウントすることで，CD メディア(700MB)という限られた領域に約 2GB のコンテンツを格納可能にする．

1.1. cloop とは

cloop(compressed loopback device)とは，圧縮ループバックデバイスの事で，圧縮ブロック伸張機能を付加したループバックデバイスである．KNOPPIX では通常 64KB 単位でルートイメージを圧縮し，cloop イメージファイルを作成する．ただし，圧縮後の cloop ブロックはシーケンシャルに配置され，ブロックサイズの変化に対応できない為，イメージ作成後には変更ができない読み込み専用ブロックデバイスとなっている．また，cloop 上に構成されるファイルシステムは何でも良く，iso9660，ext2 などが用いられている．

ライブ CD として有名な KNOPPIX だが，そのブートデバイスは CD だけでなく，DVD，USB メモリなどの CF 系，ネットワークなど実に幅広いものとなっている．特に HTTP-FUSE KNOPPIX[6]では，分割圧縮ブロックファイルによるループバックデバイス(HTTP-FUSE-CLOOP)を利用しており，cloop ブロックをファイルとして扱い，必要な部分を逐

一 Web サーバからダウンロードするネットワーク透過な環境を実現している．

いずれの場合も機能的にはCD版のKNOPPIXと同様に圧縮ブロックデバイスが使用され，デバイスの限られた領域に多くのコンテンツを格納可能にする，ストレージとの転送速度の格差を緩和するなどの役割を果たしている．

1.2. 圧縮ブロックデバイスの問題点

幾つかの条件下において有効な圧縮ブロックデバイスだが，比較的粒度の大きいブロック単位で圧縮されており，ほんの一部の情報を参照する場合でも，その部分を含む大きな範囲を読み込み・伸張するという処理を伴うため効率的でない．必要な部分の離散状態によりその効率は決定されるが，最も負荷の掛かる処理の一つであるシステム起動では，散在する様々なファイルを参照するためブロック当たりの有効バイト率は低くなる．そこで，ファイルシステムレイヤのファイルを構成するブロックレベルで再配置可能にする事で，有効バイト率が向上し，効率的な読み込み・伸張処理が可能になると考えた．

圧縮ブロックデバイス上の最適化を行うファイルシステムは，以下の理由から Ext2 ファイルシステムとした．

○基本的に圧縮ブロックデバイス上のファイルシステムは読み込み専用であるため，耐障害性は考慮せず，軽量なファイルシステムを選択した．

○ファイル単位の先読みを行う readahead の研究[7]により，読み込み単位はファイル全体とは限らず，ライブラリなどでは一部のみの参照となることが判っている．そこで最適化機構を実装するファイルシステムは，最適化により故意にフラグメンテーションの状態を作り出せる Ext2 ファイルシステムとした．iso9660 では，ファイルシステム作成時にフラグメンテーションが全て解消される．

○Ext2 ファイルシステムでは，ファイルシステムの一部の更新がファイルシステムイメージ全体に影響しないブロックアルゴリズムを採る．そのため，分割圧縮ブロックファイルを用いる HTTP-FUSE KNOPPIX では，更新された部分を含むブロックファイルの差し替えだけでファイルシステムイメージ全体が更新されるため，アップデートが低コストとなる，派生版 KNOPPIX とのブロック共有ができるなどの利点がある．

以下，2 章で関連研究，3 章で Ext2optimizer の詳細，4 章で実装，5 章で評価，6 章でまとめと今後の展開について述べる．

2. 関連研究

圧縮ブロックデバイスから効率的に情報を取得するために，ファイルシステムイメージ作成後にイメージファイルに対し最適化処理を施す関連研究として，e2defrag[8]，Ext2kai[9]，LCAT[10]を挙げる．

2.1 e2defrag

これは，一般的にフラグメンテーションを起こしたファイルは参照に時間が掛かるため，ファイルを構成するブロックを連続配置し直すことで，読込時間の効率化を図る．

圧縮ファイル作成時は，作成したいルートファイルシステムの総容量より少し大きめのループバックファイルに，ファイルをコピーするため，コピー処理終盤になると連続した十分な領域を確保できないため，意図しないフラグメンテーションが頻発し，ファイルがイメージ全体に散在する．e2defrag によりデフラグされたファイルは，ファイル参照時の圧縮ブロック数を抑えることを可能にする．

2.2 Ext2kai

これは，ext2 ファイルシステムのアルゴリズムをライトワンスメディアに特化した書き込みアルゴリズムに変更した専用ファイルシステムである．

Ext2kai は，ハードディスクの様な書き換え可能デバイスを想定した Ext2 ファイルシステムのアルゴリズムを改変し，ファイルを指定順に，断片化無く，スパース無しで配置できる．そのため，ランダムアクセスの不得手なデバイスでも効率的な読み込みが可能になる．

Ext2optimizer でも指定順にスパース無しで配置できるが，その単位は，ファイルを構成するブロック単位であり，また，場合によっては故意にフラグメンテーションを生成させる点が大きく異なる．

2.3 LCAT

これは，ランダムアクセスが不得手な CD ドライブの読み込み速度の遅さを，過去の参照情報を基に，cloop ブロックをシーケンシャルに配置し CD ドライブのピックアップの移動を最小限に抑える(以後 cloop 最適化)ことで解消する．Accelerated-KNOPPIX は LCAT により高速化された派生版 KNOPPIX である．

Ext2optimizer も同様の手法で過去の参照情報を基に最適化を行うが，適用レイヤがファイルシステムであること，より粒度の細かい最適化が行える点が異なる．また，これらの最適化手法は適用レイヤが異なるため排他的な技術ではなく，それぞれ最適化を行うことで相乗効果も期待できる．

3. Ext2optimizer の詳細

図 1 に Ext2optimizer によるファイルシステムイメージ最適化の流れを示す．Ext2optimizer は既存の Ext2 ファイルシステムから新たに最適化

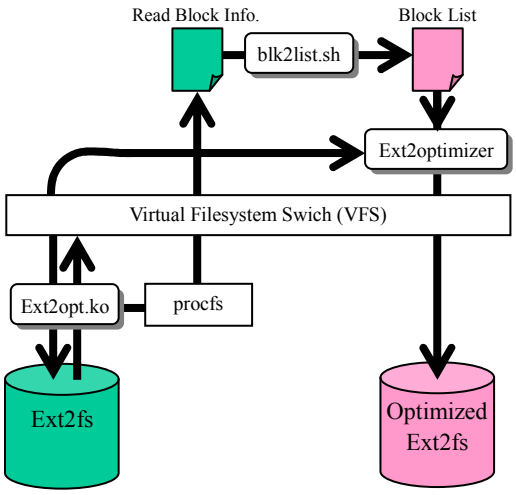

図 1. Ext2optimizer による最適化の流れ

された Ext2 ファイルシステムを生成する(以後 Ext2 最適化)．最適化の手順は，以下の通りである．

○要求ブロック情報の取得

VFS を経由してそれぞれのファイルストアに渡される要求ブロック番号を取得する．Ext2 ファイルシステムでは，ファイルのメタ情報を i-node に格納しており，そのデータブロックは 12 個の直接参照と 3 個の間接参照(一次，二次，三次)で管理している．要求されたデータブロックが間接参照されていた場合，目的のデータブロックを参照するには，予め参照経緯順に間接ブロックを読み込んでおく必要があるため，Ext2.ko にプロファイラを組み込んだ Ext2opt.ko(以後 Ext2 プロファイラ)は間接ブロックを含む形でブロック参照情報を蓄積する．

希望する全てのブロック参照情報を蓄積した後，その情報を procfs 経由で取得する．取得したいデバイスが複数の場合，デバイス毎にそれぞれ情報を取得する．

○ブロック参照情報のフォーマット変更

Ext2 プロファイラで取得したブロック参照情報を基に間接ブロックも含めたデータブロック

を重複無くシーケンシャルに配置するためのリストを作成する.

○Ext2 ファイルシステムの最適化処理

前項で作成したリストと最適化したいファイルシステムを引数に ext2optimizer コマンドを実行することで, データブロックがリスト順に詰めて配置されたイメージファイルを出力する. このイメージファイルを圧縮ファイル化することで, 希望する圧縮ブロックの効率的な読込・伸張が可能になる.

4. 実装

Ext2optimizer と各種ツールの実装, 及び Ext2optimizer により最適化されたファイルシステムを組み込んだライブ CD について述べる.

4.1 Ext2opt カーネルモジュール

Ext2 プロファイラは, カーネル空間でブロック参照情報を蓄積し, procfs 経由でユーザに情報を提供する. Ext2 ファイルシステムは, カーネルモジュールとしてではなく, カーネルに組み込まれていることが多く, また, Ext2 ファイルシステムはソースの改変が比較的穏やかであるため, Ext2 の patch として linux-2.6.11 (linux-2.6.12)に実装した.

4.2 blk2list.sh フィルタコマンド

procfs 経由で取得した Ext2 ブロック参照情報はそのデータブロックがどの間接ブロックを経由して参照されているかを時系列に取得した情報である. Ext2 最適化を行うには, 冗長な情報であるため, ユニークなブロックリスト(間接ブロックを含む)を出力する.

4.3 Ext2optimizer コマンド

Ext2optimizer は 3 つの行程により, 最適化された Ext2 イメージファイルの出力を行う.

① 初期化処理

まず, 対象となるファイルシステム, 4.2 で作成した最適化リストファイルを次の行程のためにメモリに格納する. 一般的に対象となるファイルシステムは数 GB になる場合が少なくない. そのため, ここで読み込むのは, Ext2 ファイルシステムの各ブロックグループのメタブロック(スーパーブロック, グループディスクリプタ, データブロックビットマップ, i ノードビットマップ, i ノードテーブル)と, 間接ブロックなどの制御情報のみである.

2GB 程度のファイルシステムの場合, およそ 50 万ブロックを操作することもあるため, これらの情報を AVL 平衡木で管理し, 全てのブロックへのアクセスを O(logN)に抑える.

② 最適化処理

最適化対象となるファイルシステムは, 予め Ext2 ファイルシステムアルゴリズムにより, ディレクトリ単位で関連するファイルが可能な限り集約配置されている. そのため, 最適化処理は, 以下に示す単純明快なアルゴリズムで動作し, 必要となるブロック以外は可能な限りその状態を維持する. この処理での操作対象は AVL 平衡木で管理された制御情報である.

I. ファイルシステムの全てのデータブロックを後ろにずらす.
II. データブロックを最適化リスト順に先頭から詰める. I.の先頭ブロックと重なる時は, 残りのブロックを全て後ろにずらす.
III. 最後に残ったデータブロックを全て前にずらす.

ここで, データブロックは複数箇所から参照さ

れる場合があること(ハードリンク)，シンボリックリンクは 60Byte を超えるとデータブロックを保持することなどを考慮に入れて処理を行う．

③　ファイルシステム出力処理

最後は，メモリ上に保持する制御情報が最適化済みのファイルシステムイメージとなるため，Ext2 ブロック番号順に標準出力する．メタブロックや間接ブロックはメモリから，データブロックは最適化対象となるファイルシステムをデバイスから読み込み，それぞれ出力する．

4.4 KNOPPIX への適用

Ext2 プロファイラを組み込んだ linux-2.6.11 をライブ CD の KNOPPIX3.8.2 日本語版(以後 A-①)に組み込み，VMware5.0(build-13124)にて Ext2 ブロック参照情報を取得し，Ext2optimizer で最適化したファイルシステムイメージを作成し，そのイメージから KNOPPIX(A-②)を作成した．さらに，LCAT を用いた cloop 最適化とブロック先読み機能を，A-①，A-②にそれぞれ適用した(A-③，A-④)．cloop ブロック参照情報は Ext2 ブロック参照情報と同じく VMware5.0 で取得した．作成したライブ CD を表 1 に示す．

表 1 作成したライブ CD

ID	説　　明
A-①	KNOPPIX3.8.2 日本語版
A-②	Ext2 最適化
A-③	cloop 最適化
A-④	Ext2 最適化 ＋cloop 最適化

4.5 HTTP-FUSE KNOPPIX への適用

KNOPPIX4.02 日本語版のファイルシステムイメージを Ext2 最適化したものとそうでないものをそれぞれ圧縮ブロックファイルに変換して，リモートの Web サーバに配置した．分割サイズは 64，128，256，512KB の 4 種類である．作成したライブ CD は HTTP-FUSE KNOPPIX-4.0.2 に Ext2 プロファイラと cloop プロファイラを組み込み，プロファイラによるデータ取得はオプションにより選択できる．つまり，この一枚で Web サーバに設置した計 8 パターンの環境で起動できる(表 2)．

表 2 作成した HTTP-FUSE KNOPPIX 環境

Block Size	ファイルシステムイメージ	
	Normal	optimize
64	B-①	B-⑤
128	B-②	B-⑥
256	B-③	B-⑦
512	B-④	B-⑧

5. 評価

前章で実装したライブ CD(KNOPPIX，HTTP-FUSE KNOPPIX)について評価した．評価は全て boot プロンプトから KDE スプラッシュが消えるまでのシステム起動について行った．

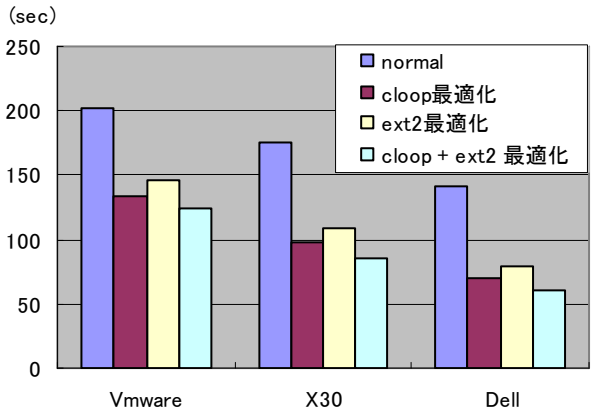

図2. 起動時間の変化

表 3 性能評価クライアント PC 仕様

VMware(Dell Evo Desktop D530SF)	
CPU	Pentium4 3.2GHz(HT)
MEM	384MB
CD-Drive	40 倍速
IBM ThinkPAD X30	
CPU	PentiumIII-M　1.06GHz
MEM	768MB
CD-Drive	24 倍速(ドッキングステーション)
Dell Evo Desktop D530SF	
CPU	Pentium4 3.2GHz(HT)
MEM	1.5GB
CD-Drive	40 倍速

5.1 KNOPPIX の評価

4.4 で実装したライブ CD について計測した起動時間を図 2 に示す．また，計測に利用した PC スペックは表 3 である．

A-③では，cloop 最適化により CD 起動のボトルネックであったシーク時間が削減され，既存の起動時間の約半分で起動しており，ライブ CD のボトルネックの大半は CD ドライブのシーク時間が問題であったことが伺える．

A-②では，ファイルシステムが最適化されたので読み込まれる cloop ブロック数が約 60%になった．Ext2 最適化によりブロック数が削減でき，更に Ext2 ブロックがシーケンシャルに配置されたため，結果的に cloop ブロックもシーケンシャルに配置され，A-③と同様にシーク時間が削減された．しかし，Ext2optimizer により最適化されるのはデータブロックのみであり，i ノードなどのメタ情報を保持する cloop ブロックはシーケンシャルに配置できないため，A-③より若干遅い結果となった．A-④では，A-③でシーケンシャルにできなかった i ノードを保持する cloop ブロックもシーケンシャルに配置できたため，更に 10 秒程の高速起動を計測できた．cloop と Ext2 は実装レイヤが異なるため，それぞれの最適化処理を打ち消し合うことなく相性が良い結果となった．

表 4 性能評価 Web サーバ PC 仕様

Dell Precision WorkStation	
CPU	Pentium4　2GHz
MEM	512MB
kernel	2.6.11
Apache	1.3.33-8

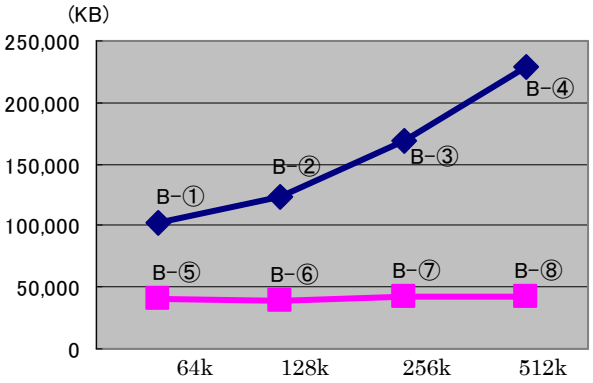

図3．総ダウンロードファイルサイズ

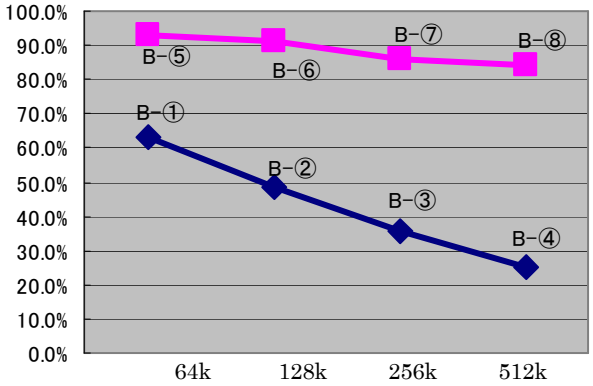

図4．有効バイト率

5.2 HTTP-FUSE KNOPPIX の評価

4.5 で実装した HTTP-FUSE KNOPPIX について計測を行った．計測に利用したクライアント PC は表 3 の X30，圧縮ブロックファイルを配信する Web サーバは，表 4 である．

図 3 はシステム起動に必要な圧縮ブロックファイルの総ダウンロードサイズを表す．B-①～④はファイルシステムが最適化されておらず，切り出しブロックサイズにより指数関数的にダウンロードサイズが増加している．それに対し，B-⑤～⑧では，ファイルシステムが最適化されているため，分割サイズの変化によらずダウンロードする量はほぼ横ばいである．公開されている HTTP-FUSE KNOPPIX では分割圧縮ブロックファイルを論文[5]で最適とされた 256KB 単位で作成している．

表 5 に B-①～⑧のシステム起動に必要な cloop ブロック，Ext2 ブロックの計測値，図 4 にダウンロードした cloop の内，実際に使用された Ext2 ブロックの割合を示す．ファイルシステムが最適化されていない B-①～④では，切り出しサイズが大きくなるにつれ，伸張された cloop ブロック

表 5 cloop ブロックサイズと Ext2 最適化による各ブロック数の変化

cloop ブロックサイズ			normal	optimize
64k	cloop ブロック	ユニークな cloop ブロック数	2,554	1709
		伸張後のサイズ(KB)	163,456	109,376
	ext2 ブロック (blocksize 4KB)	ユニークな ext2 ブロック数	25,863	25,359
		サイズ(KB)	103,452	101,436
	有効バイト率(%)	伸張した cloop の有効バイト率	63.3%	92.7%
128k	cloop ブロック	ユニークな cloop ブロック数	1,658	870
		伸張後のサイズ(KB)	212,224	111,360
	ext2 ブロック (blocksize 4KB)	ユニークな ext2 ブロック数	25,817	25,359
		サイズ(KB)	103,268	101,436
	有効バイト率(%)	伸張した cloop の有効バイト率	48.7%	91.1%
256k	cloop ブロック	ユニークな cloop ブロック数	1,125	461
		伸張後のサイズ(KB)	288,000	118,016
	ext2 ブロック (blocksize 4KB)	ユニークな ext2 ブロック数	25,817	25,359
		サイズ(KB)	103,268	101,436
	有効バイト率(%)	伸張した cloop の有効バイト率	35.9%	86.0%
512k	cloop ブロック	ユニークな cloop ブロック数	802	240
		伸張後のサイズ(KB)	410,624	122,880
	ext2 ブロック (blocksize 4KB)	ユニークな ext2 ブロック数	25,817	25,806
		サイズ(KB)	103,268	103,224
	有効バイト率(%)	伸張した cloop の有効バイト率	25.1%	84.0%

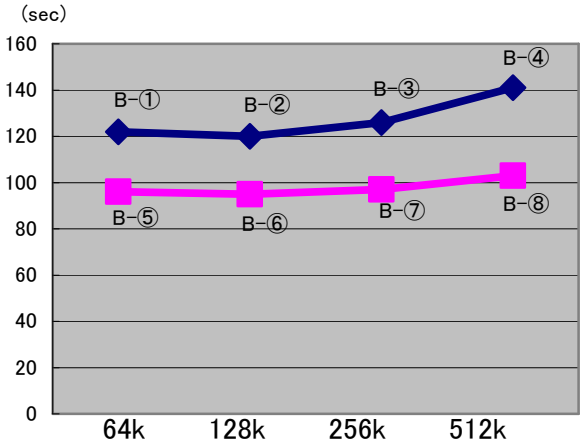

図5. delayなしの起動時間

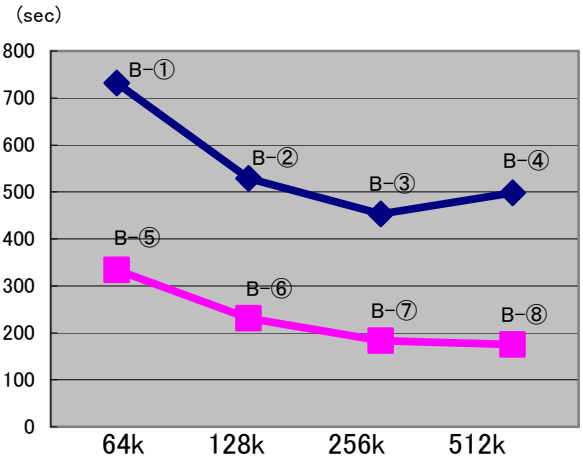

図6. 100ms delayの起動時間

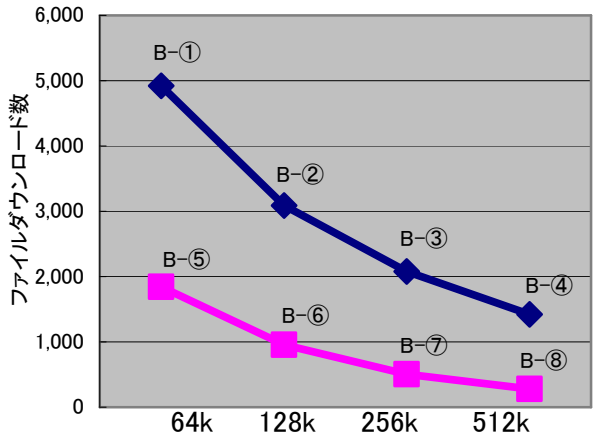

図7. Webサーバのファイルダウンロード数

で実際に使われる Ext2 ブロックはごく僅かであることが判る．それに対し B-⑤～⑧では，ほぼ横ばいであり，切り出しブロックサイズの影響を受けにくい．有効バイト率が 100%でないのは，Ext2optimizer が最適化するのはデータブロックのみであり，i ノードを含むメタデータは最適化されないためである．

切り出しブロックサイズと最適化の有無による起動時間を図 5(delay なし(0.1ms))，図 6(100ms delay) に示す．ネットワーク遅延は，Web サーバ側で netem (kernel-2.6.11)[11]によ

り発生させ，コネクション数は 1 つとした．ネットワーク遅延がない場合は 20～40sec 程であった起動時間が，ネットワーク遅延(100ms)が大きくなるとその格差がより顕著に表れた．B-⑤～⑧ではダウンロードする cloop ブロックファイルの絶対数が減少したため，レイテンシの影響を受けにくいことが判る．また，Web サーバで保持するログを解析し，GET request をカウントしたところ図 7 のようになった．Ext2 最適化によりダウンロードされるファイル数は最大 80%削減されるため，サーバの負荷，ネットワークトラフィックを軽減でき，スケーラビリティ向上に繋がると言える．

6. まとめと今後の展開

本稿では，ファイルシステムレイヤのファイルを構成するブロックレベルで最適化する Ext2optimizer についての詳細とそれにより最適化されたファイルシステムを KNOPPIX と HTTP-FUSE KNOPPIX にそれぞれ実装し，システム起動について簡単な評価を行った．その結果，システム起動に必要な cloop ブロック数を抑制し，効率的な起動ができた．

Accelerated-KNOPPIX の高速起動の主要となる手法は，cloop ブロックをシーケンシャルに配置し，そのブロックを先読みすることである．それに対し，Ext2optimizer は cloop 上のファイルシステムの最適化により cloop ブロック当たりの有効バイト率を向上させ，cloop ブロックの絶対数を減少させる．これらは実装レイヤが異なるため，互いの欠点を補完でき，更に約 10 秒の高速化が計測できた．

HTTP-FUSE KNOPPIX では，Ext2 最適化による cloop ブロックファイルの低減によりレイテンシの影響を受けにくくする効果が十分に計測された．現在公開されている圧縮ブロックファイルは，レイテンシと有効バイト率とのトレードオフの関係から，圧縮ブロックサイズを 256KB としているが，Ext2 最適化によりダウンロード総量を 75%以上削減できたことは，ネットワーク起動の高速化に繋がるだけでなく，次回起動用に保持するキャッシュの低容量化を実現する．具体的には圧縮ブロックサイズが 256KB であった場合，システム起動のために保持するキャッシュは約 170MB 必要であるが，Ext2 最適化により，僅か 40MB のキャッシュを保持するだけでよい．

また，ダウンロードファイルの減少により，Web サーバに対する負荷軽減，ネットワークトラフィックの減少などネットワークリソースの有効利用が期待できる．

今後，Ext2optimizer に i ノードなどの制御情報の最適化機構を組み込むことで更なる最適化を見込める．また，Ext2optimizer の適用範囲を広げ，デフラグ的な最適化手順を踏むことで，起動に最適化されたハードディスクイメージ作成が可能になると考えられる．

本研究で実装した Ext2optimizer は，ある時点でソースコードを広く公開し，今後も開発を進めていく予定である．

参考文献&URL

[1] KlausKnopper, ”Building a self-contained auto-configuring Linux system on an iso9660 filesystem”, 4th Annual Linux Showcase & Conference,Atlanta http://www.usenix.org/publications/library/proceedings/als00/2000papers/papers/full_papers/knopper/knopper.pdf
[2] squashfs, http://squashfs.sourceforge.net/
[3] knoppix, http://www.knopper.net/knoppix/
[4] Accelerated-KNOPPIX, http://www.alpha.co.jp/ac-knoppix/
[5] KNOPPIX 日本語版 http://unit.aist.go.jp/itri/knoppix/index.html
[6] 須崎,八木,飯島,丹,”HTTP-FUSE KNOPPIX”, Linux Conference2005, http://lc.linux.or.jp/paper/lc2005/CP-02.pdf
[7] KNOPPIX3.7 日本語版(IPA フォント+readahead 付き) http://unit.aist.go.jp/itri/knoppix/news/news050201.html
[8] e2defrag, http://e2compr.sourceforge.net/defrag.html
[9] 北川,丹,千葉,須崎,飯島,八木,” ライブ CD へ向けた Ext2 ファイルシステムの書込みアルゴリズムの検討と評価” http://www.alpha.co.jp/knoppix/download/paper/FIT2005_9_paper.pdf
[10] LCAT, http://sourceforge.jp/projects/lcat/
[11] netem, http://linux-net.osdl.org/index.php/Netem