ネットワークに対応した分割圧縮ループバックデバイス HTTP-FUSE-CLOOP とそれから起動する Linux

http://unit.aist.go.jp/itri/knoppix

須崎有康 [1)], 八木豊志樹 [1)], 飯島賢吾 [1)], 北川健司 [2)], 田代秀一 [1)]

産業技術総合研究所 [1)], ㈱アルファシステムズ [2)]

概要：ブロックデバイスを分割圧縮したブロックファイルから再構成できるループバックデバイス HTTP-FUSE-CLOOP を開発した。分割圧縮したブロックファイルを手元の二次記憶からでも HTTP 経由でリモートからでも透過的に扱える。この HTTP-FUSE-CLOOP を使ってルートファイルシステムを Internet から mount して起動する Linux “HTTP-FUSE KNOPPIX”を開発した。この設計の方針と性能について述べる。

Network compressed loopback device “HTTP-FUSE-CLOOP” and Linux which boot form it.

Kuniyasu Suzaki[1)], Toshiki Yagi[1)], Kengo Iijima[1)], Kenji Kitagawa[2)], Shuichi Tashiro[1)]

National Institute of Advanced Industrial Science and Technology [1)], Alpha Systems Inc. [2)]

Abstract: We developed network compressed loopback device “HTTP-FUSE-COOP” which can re-construct block device from the split-compressed block files. The split-compressed block files are transparently treated between local storage and remote network. We developed a Linux “HTTP-FUSE KNOPPIX” which mounts root file system using HTTP-FUSE-CLOOP via Internet. In this paper we describe the detail of implementation and its performance.

1. はじめに

我々は Internet 上でルートファイルシステムを公開し、そこからブートするクライアント OS の研究を行っている。不特定多数のユーザを対象に、世界中のどこからでも Internet さえ接続できればハードディスク不要で利用可能な OS の開発を目指している。

Internetでルートファイルシステムを公開する方法にはブロックデバイスレベルの iSCSI[1]やファイルシステムレベルの NFS4[2], Open-AFS[3], SFS[4], SHFS[5]などが考えられる。しかし、これらは専用ポートを想定しているためファイアウォールを越えた利用が不可能な場合がある、不特定多数の利用を考慮していない、常時接続を前提としている、純粋なサーバクライアントモデルで拡張性がない、などの問題点を持つ。我々はこれらの問題点を解決するため、細かく分割し、圧縮されたブロックファイルをまとめて１つのループバックデバイスに見せる HTTP-FUSE-CLOOP を開発した。

分割圧縮ブロックファイルは、アクセスがあった時にオンデマンドで HTTP サーバからダウンロードされる。ダウンロードされたファイルは二次記憶に保持し、再度アクセスがあった場合にはそこから利用する。つまり、ネットワーク透明であり、二次記憶に必要な分割圧縮ブロックファイルがあればネットワークの接続が切れても構わない。分割圧縮ブロックファイルはダウンロードさえできればよく、HTTP の場合では Proxy キャッシュを利用してダウンロードサーバへの負荷集中を避けると共に短いレンテンシを実現することができる。

また、分割圧縮ブロックファイルへのアクセスに物理的な番地を利用するのではなく、ブロックコンテンツの内容によるハッシュ値(MD5)による検索にした。この検索のためにハッシュ値のインデックスファイルを用意した。これにより空のデータを含むブロックや同一内容のブロックは１つの分割圧縮ブロックファイルにまとめられ、領域を削減できる。ハッシュ値によりブロックの内容の確認ができるため、セキュリティも向上できる。内容更新はハッシュ値の違う分割圧縮ブロックファイルを追加し、インデックスファイルを更新するのみで実現できる。もとの分割圧縮ファイルを消去する必要がないため以前の状態に戻すことも可能である。このようなブロックレベルでハッシュ値による検索は Plan9 の Venti[6], CFS[7], Bittorrent[8], BTSlave[9] などでも使われている手法である。これらとの比較は

2 章で詳細を述べる。

開発した HTTP-FUSE-CLOOP は 1CD/1DVD Linux である”KNOPPIX”[10][11]と組み合わせ、Internetからルートファイルシステムを取得して起動する OS である”HTTP-FUSE KNOPPIX”[12]を開発した。最新版の KNOPPIX4.0 は使用するメディアが CD から DVD へと拡張されて容量が 3.8GB となり、ダウンロードも容易ではなくなってきている。HTTP-FUSE KNOPPIX では、機能的には通常の DVD 版 KNOPPIX と同じであるが、DVD 版のように全てのルートファイルシステムをダウンロードして持ち歩く必要はない。現在の HTTP-FUSE KNOPPIX のブートローダ部分の iso イメージは 5MB 程度なので、このイメージを CD に焼いたものでインターネットからブートすれば DVD 版 KNOPPIX と同様な利用ができる。また、アプリケーションが更新されても、サーバ上のルートファイルシステムを変えるだけで新たに DVD を作成する必要はない。

以下、2 章で関連研究との比較、3 章で HTTP-FUSE-CLOOP の詳細、4 章で性能評価について報告する。5 章で HTTP-FUSE KNOPPIX が持つ問題点を議論し、6 章で今後の課題、7 章でまとめを述べる。

2. 関連研究

2.1. Plan9 の Venti

Plan9 用に開発された archival block storage server の”Venti[6]”は、データの読み書きをデバイス固有のアドレスではなく、書き込むブロックの中身についてユニークなハッシュ識別子をキーとして行う。システムのインタフェースはブロックの読み書きを許す簡単なプロトコルであり、ファイルシステムの機能は archival file server である”fossil”などが受け持つ。Venti はバックエンド的な長期データ保存ストレージである。

HTTP-FUSE KNOPPIX でもブロック内容の MD5 ハッシュ値をキーとするストレージ管理が似ているが、ブロックの保存先がファイルである点が大きく異なる。ファイルにすることで HTTP 配信が可能となり、Venti で用意している特殊なプロトコルやソフトウェアは不要である。また、ファイル形式ならコピーも簡単に作成でき、proxy によるファイルキャッシュを使った Internet 配信を活用することができる。

2.2. CFS: Cooperative File System

CFS[7]は MIT で開発されている分散ハッシュ chord ベースの P2P ファイルシステムである。CFS はファイルが格納されているページを P2P のノードに分散配置する。ページの検索には chord を用いる。CFS ではノードを追加してもブロックの再配置は即座に行われないため、高速に複製を増やし、高バンド幅、低レイテンシを実現するファイルシステムではない。CFS は実用よりも実験的実装な意味合いが強く、信頼性、拡張性に主眼が置かれている。

2.3. Bittorrent と BTSlave

Bittorrent[8]は、大規模ファイルに適した高速 P2P ダウンロードソフトウェアである。Bittorrent では大規模ファイルを細かいピースと呼ばれるデータに分割して、その細かいピースを複数の P2P ノードから「バンド幅が太くなる戦略で」ダウンロードする。この戦略が Bittorent を特長付けるもので、下記の性質がある。

1. 複数のノードにダウンロードコネクションを張り、最終的に全てのピースが早く集まる順にダウンロードを行う。例えば、ダウンロードサイト間で保持しているピースのコピーが少ないものほど優先してダウンロードする。これはダウンロードサイトとの通信が断絶したときにそのサイトしか保持していないピースがネックにならないようにするためである。
2. ピースは全ダウンロード終了後に bencode によって元のファイルに復元する。この時までファイルは Bittorrent 管轄のピースのままであり、

ファイルとして扱うことができない。
このような戦略は大きなファイルの高速なダウンロードには適するが、ファイルのランダムな読み出し要求に適するものではない。Bittorent を元とするファイルシステムに BTSlave[9]があるが単純な拡張では高速な読み出しが難しいようでプロトコルの再実装を検討している。

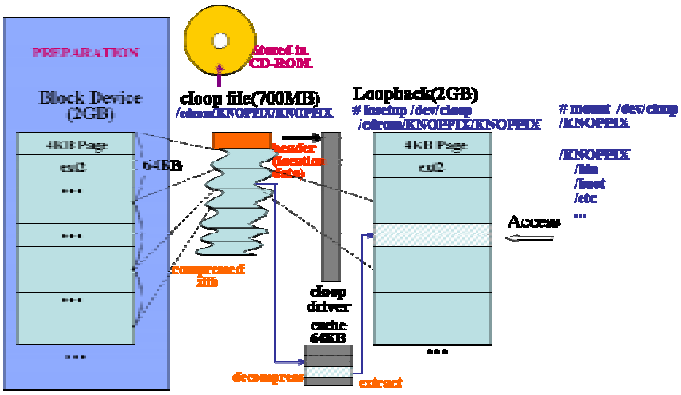

図１ KNOPPIX での cloop の利用手順

3. 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

KNOPPIX ではルートファイルシステムを圧縮ループバックデバイスに収録して１ファイルとして扱う。ループバックデバイスとはファイル内にブロックデバイスを格納できる仕組みである。KNOPPIX では CD-ROM 対応のために更に圧縮機能をつけた cloop(compressed loopback device)により容量を小さくしている。今回の開発では、この cloop を改良して、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を作成した。

3.1 cloop の仕組み

cloop ではブロックデバイスを固定サイズ(標準で 64KB)毎に切り出し zlib 圧縮して、2G程度のブロックデバイスイメージを 700MB 程度のループバックファイルに収納している。元のブロックデバイス上で使われているファイルシステム(iso, ext2, etc)に依存しない。

CD 版の KNOPPIX がブートするとき、この cloop ファイルをループバック設定してブロックデバイスとして扱えるようにする（図 1)。KNOPPIX ではブート時に CD-ROM 内の cloop ファイル /cdrom/KNOPPX/KOPPIX を/dev/cloop デバイスにセットアップしている。

```
# insmod –f /modules/cloop.o file=/cdrom/KNOPPIX/KNOPPIX
```

更に通常のファイルシステムとして読み出せるように mount 操作を行う。

```
# mount /dev/cloop /KNOPPIX
```

ここまで設定した後、cloop ファイルに格納したファイルを読み出すことができる。

この cloop 内のファイルアクセスでは、cloop デバイスドライバによるブロックの読み出しが通常の手順と異なる。cloop ではファイルの読み出し要求があると cloop driver が要求された読み出しに該当する圧縮ブロックを cloop ファイルから読み出し、伸張を行って必要な部分のみを返している。伸張した結果はデバイスドライバ内にキャッシュされ、直後に同じブロックがアクセスされた場合の高速化に役立てる（図 1)。

3.2 改良点

この圧縮ループバックデバイス cloop は、ブロックデバイスをファイルとして扱えるため便利である。しかし、常に一つの大きなファイル(CD 版では 700MB 程度)として扱わなければいけないため、ダウンロードするのに時間がかかる。

本開発では cloop ファイルを分割して複数のファイルとし、分割したファイルをリモートとローカルで透過的に扱える「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)」を開発した。改良のポイントは下記である。

- 固定サイズ(現在の標準は 256KB)毎にブロックを切り出し、圧縮したデータを”ファイル”に格納する（分割圧縮ブロックファイル)。
- 多数の分割圧縮ブロックファイルをまとめて、一つのループバックデバイスとして扱えるようにする。
- ループバックデバイスでは、必要な時に必要な分

割圧縮ブロックファイルを取得すればいいようにする。つまり、全ての分割圧縮ブロックファイルが揃わなくてもループバックデバイスとして扱えるようにする。

- 分割圧縮ブロックファイルのファイル名はその内容のハッシュ値(MD5)とし、ファイル名で内容確認ができるようにする。同一内容の場合、ハッシュ値が同じになりファイル数を減らすことができる。

今回の実装では分割圧縮ブロックファイルをまとめる手段として仮想ファイルシステム FUSE (Filesystem in Userspace)[13]を利用した(図 2)。

開発では FUSE の wrapper を利用し、FUSE を通して分割圧縮ブロックファイルを cloop ファイルとして再構成する HTTP-FUSE-CLOOP を作成した。分割圧縮ブロックファイルは cloop と同様に固定サイズでブロックを切り出し、zlib で圧縮したものをファイルに保存したものであり、ファイルの内容は cloop ファイルの分割圧縮ブロックと同一である。FUSE を通してブロックファイルを cloop として再構成するために、「cloop ヘッダの生成」と「cloop 内の分割圧縮ブロックと分割圧縮ブロックファイルの対応」を取る index.idx ファイルを作成した。cloop ファイルに対してブロック読み出し要求があると、そのブロックに該当する分割圧縮ブロックファイルを FUSE を使って取得し、要求された cloop ファイルのブロックに割り当てている(図 2)。

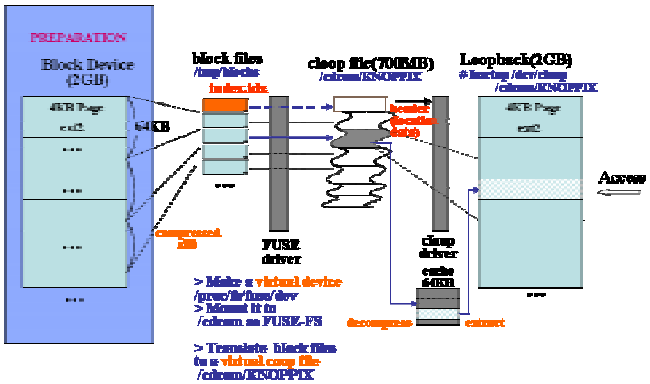

図 2 FUSE-cloop の構成

3.3. Internet 対応

分割圧縮ブロックファイルは、cloop ファイル内で該当する部分が読み出される時に検索される。そのコピーが手元の二次記憶（ハードディスクまたは USB メモリ）にあればそれを利用し、無ければ Internet よりオンデマンドでダウンロードする。オンデマンドのダウンロード部分も FUSE wrapper として libcurl を用いて作成した(図 3)。

ダウンロードした分割圧縮ブロックファイルは RAM-Disk あるいは二次記憶に保存される。二次記憶に保存された場合、2 回目以降のブートでは保存された分割圧縮ブロックファイルを利用する。

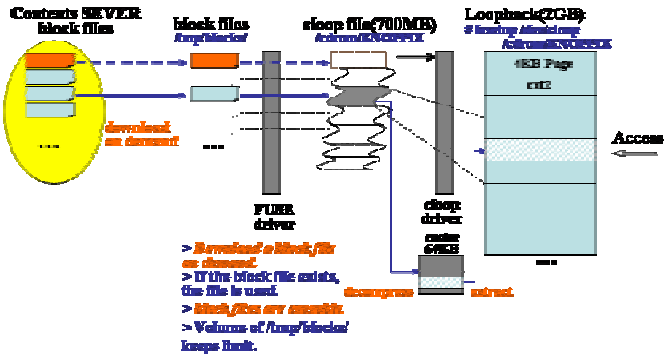

図 3 HTTP FUSE-cloop の構成

3.4. 差分更新

分割圧縮ブロックデバイスの利点として、アプリケーションの更新が起こった場合、その差分部分と index.idx ファイルの入れ替えのみで実現できる（図 4)。この機能は、元のブロックデバイス上のファイルシステムが ext2 のようにブロック単位で更新可能なことを条件とする。iso9660 のように一部の変更が他のブロックの位置まで変えるようなファイルシステムを適用している場合にはこの機能が利用できない。

今までの CD 版の KNOPPIX では、元のファイルシステムが ext2 としても cloop ファイルにする時に、一部の変更がそれ以降のブロック配置に影響を与えていたために差分更新することができなかった。HTTP-FUSE-CLOOP では、ブロックをファイルとして扱えるようにしたために、変更の生じたファイルの変更とその位置情報である index.idx の

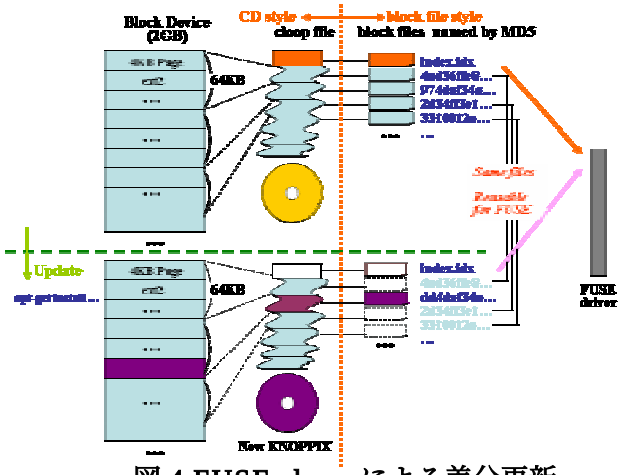

図 4 FUSE-cloop による差分更新

みの変更で差分更新に対応可能となった。

追加するファイル名は更新したブロックの内容の MD5 値となる。HTTP-FUSE-CLOOP では、新しい index.idx と新たな分割圧縮ブロックファイルを追加すれば以前の分割圧縮ブロックファイルを再利用することができる。これにより、カスタマイズを行っても更新差分のファイルのみを提供すればよいので、効率的な配布を行うこと可能となる。

差分更新を利用したサーバとクライアントの構成は図 5 になる。サーバ側では、同一ブロックファイルをシンボリックリンクで共有することで容量の削減が可能となる。また、クライアントではブート時にルートファイルシステムを切り替えることが可能で、１つの HTTP-FUSE KNOPPIX のブート用 CD-ROM から多くの KNOPPIX カスタマイズが利用できる。

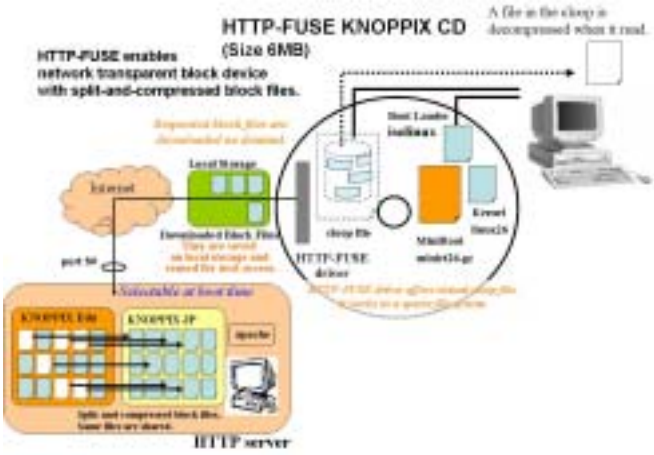

図 5 HTTP-FUSE KNOPPIX の構成

3.5. 欠点と対処(netselect と DLAHEAD)

HTTP-FUSE-CLOOP を使った場合のダウンロード要求は下記の手順になる。

ext2 -> cloop -> FUSE -> *(HTTP Internet)* -> file

cloop は仮想ブロックデバイスのためアクセス要求が逐次化される。これを Internet に適応した場合、コネクションが１つになってしまう。これでは NFS などで一般に使われているネットワークのバンド幅拡張の手段が適用できない。つまり、バンド幅拡張はクライアント・サーバ間に複数のコネクションを張り、ネットワークのレイテンシを隠蔽する手法であるが、その手法が適用できない。特に何も手がかりのないブート時にこの影響を受けやすい。

この問題点を解消するために、ブート時に短いレイテンシのサーバを見つける機構(netselect)と事前に解析した必要ブロックファイルを平行ダウンロードする機構(DLAHEAD)をブートシーケンスに組み込んだ。

3.5.1 netselect

HTTP-FUSE-CLOOP を使った Internet からのブートでは、クライアントがレイテンシを短くするようにダウンロード先を選択する機構を加えた。具体的には複数の HTTP サーバを用意し、その IP アドレスに netselect をかけてレイテンシの短いものを見つける。netselect は traceroute を使ってクライアントから近いサーバを見つけるツールである。HTTP-FUSE KNOPPIX のブートでは一番短いサーバからダウンロードすることで高速なブートを実現する。

3.5.2 DLAHEAD

ブート時に必要なファイルを事前に解析し、必要と思われるブロックを HTTP-FUSE-CLOOP と平行してダウンロードする DLAHEAD(download ahead) の機能を付けた。DLAHEAD では、サーバ側に置かれたブロックファイルリストを見つけると、複数のコネクション（デフォルトで４本）を張って、事前に必要と思われるファイルをダウンロードする。ダウンロードしたファイルは二次記憶に保存される。HTTP-FUSE-CLOOP から要求があった際

にダウンロード済みであれば二次記憶のファイルを利用する。

ただし、この方法でも DLAHEAD のリストに該当しない読み出し要求があった場合（異なるデバイスドライバやブートオプションなど）は、逐次にダウンロードしなければならない問題点をもつ。

4. 性能測定

4.1. HTTP-FUSE-CLOOP の性能

まず、HTTP-FUSE-CLOOP 単体の性能を測定した。KNOPPIX4.0 DVD 版を元に分割圧縮ブロックファイルを作成した。切り出しサイズは論文[12]で最適とされた 256KB とした。結果は表 1 になる。

表 1 256KB 分割圧縮ブロックファイルの特徴

元の cloop ファイル	ファイル数	ファイルの総容量	ファイルのサイズ
3.15GB (展開後は 9.7GB)	35,504	3.09GB	Max: 2,622,230 Min: 277 Ave: 93,581

元になった cloop ファイルは 64KB ごとに圧縮をかけている。256KB ごとにすることで若干圧縮率が高まり、総容量が低減できた。また、平均ファイルサイズはネットワークの Window サイズの 64KB に乗るようになっている。これ以下の切り出しでは Window サイズを余らせ、バンド幅が小さくなってしまう。輻輳制御のためのスロースタートの影響も考えられるが、TCP コネクションは keep-alive を有効にしており、ウインドウサイズに近いデータのため影響は少ない。

この分割圧縮ブロックファイルを用いて HTTP-FUSE-CLOOP の性能を測定した。利用したマシン仕様は表 2 である。

表 2 性能評価マシン仕様

HTTP-Server:	HTTP-FUSE Client:
IBM ThinkPAD T24 CPU Pentium III 1GHz, メモリ 1GB, 100M NIC apache 1.3	IBM ThinkPAD T42 CPU Pentium M 1.8GHz, メモリ 2GB, 1000M NIC 24 倍速 CD ドライブ

レイテンシの影響を調べるため、FreeBSD の dummynet によるレイテンシ(往復 200msec。日米間で片道 100msec を想定)も入れて測定した。測定では「dd によりブロックレベルの読み出し」と「tar によるファイルシステムレベルでの読み出し」をそれぞれ求めた。比較のために二次記憶(ハードディスク)にブロックファイルを置いた場合の性能も測定した。参考のために元になった cloop ファイルとそれの分割圧縮ブロックファイルのダウンロードによるスループットも求めた。この結果は表 3 になる。

表 3 HTTP-FUSE-CLOOP の性能

	delay なし	200msec delay
HTTP からの dd	2.51 MBps	0.194 MBps
HTTP からの tar	6.41 MBps	0.516 MBps
HD からの dd	15.4 MBps	
HD からの tar	20.4 MBps	
(参考性能)		
cloop ファイルの wget ダウンロード	9.56 MBps	0.320 MBps
ブロックファイルの wget ダウンロード	2.71MBps	0.197 MBps

この結果から HTTP-FUSE-CLOOP はレイテンシの影響を受けやすいことが判る。これは小さな分割圧縮ブロックファイルを直接ダウンロードした場合のバンド幅とほぼ同じ性能であり、細かいファイルを逐次にダウンロードすることが大きく影響する。cloop ファイルのような大規模ファイルではレイテンシが短い場合は NIC の性能に近いバンド幅を出せるが、分割圧縮ブロックファイルではレイテンシが短くても、オーバーヘッドの影響でバンド幅が稼げない。

ファイルシステムを通して tar した場合は、圧縮の効果が現れており、レイテンシのあるなしに関わらず、dd のブロックアクセスより約 2.5 倍のバンド幅に拡張されている。現在の CPU は性能が向上しているため、ネットワークの性能を隠蔽する良い手段であることが判る。

4.2. HTTP-FUSE KNOPPIX のブート

次に HTTP-FUSE KNOPPIX のブートについて測定した。分割圧縮ブロックファイルやマシン条件は同じである。

事前調査としてブート時にダウンロードされるファイルを調べた。ファイル数は 3,107 であり、そ

の容量は 234MB であった。これは DLAHEAD で使われるファイルリスト(DL-FULL)になる。残念ながらこのリストを DLAHEAD の標準にするとメモリが 512MB のマシンではブート途中でキャッシュが溢れ、ブートできないことが判っている。このため、メモリが 512MB でも動くように最初の 2500 ファイルとしたリスト(DL-2500)も用意した。この場合の総容量は 186MB である。DLAHEAD のダウンロードコネクションはデフォルトの４本とした。測定ではそれぞれの DLAHEAD を行う場合、行わない場合についてレイテンシを変えて評価した。

4.2.1 起動時間

起動時間の測定では boot: プロンプトからルートファイルシステムをマウントするまでの時間(T-root)、それ以降のブロックファイルのダウンロード時間(T-blcok)、最終的にデスクトップマネージャの KDE が終了するまでの時間(T-KDE)を測定した。ブートの所要時間を表５にまとめる。

表 5 ブートの所要時間(単位 秒)

		ルートファイルシステムを要求するまでの時間 (T-root)	ブロックファイルのダウンロード時間 (T-block)	KDE までの起動時間 (T-KDE)
latency なし	DL FULL	37	23	97
	DL 2500		61	98
	DL なし		76	113
latency 200msec	DL FULL		402	437
	DL 2500		607	644
	DL なし		1376	1413
DVD 起動(参考性能)		22	273	295

T-root は CD や DVD からのブート処理で消費される時間である。HTTP-FUSE KNOPPIX が通常の DVD 起動より時間が要するのは、T-root の時間内でネットワークの設定や HTTP-FUFE の準備を行うためである。

基本的に T-KDE は T-root と T-block の和であるが、latency なしで DL-FULL の場合は先行ダウンロードが早めに終了して HTTP-FUSE-CLOOP の本体は二次記憶のキャッシュを利用するため T-block(23sec)が短くなっている。しかし、T-KDE (97sec)を比較すると DL-2500(98sec)とほとんど変わらない。つまり、ある程度先行ダウンロードを行えば、ブート処理で律速されていることがわかる。全体を逐次にダウンロードにした場合は、ダウンロードに要する時間が隠蔽できず、T-KDE (113sec)も遅くなる。いずれにしても latency がない場合は DVD 起動より高速であった。

latency が 200msec の場合は、DLAHEAD ありでもダウンロード時間を隠蔽できず、いずれも DVD 起動より遅くなってしまう。しかし、DLAHEAD なし(1,413sec)と比べると 2 倍以上起動時間短縮(437sec と 644sec)が確認できる。

4.2.2 転送データ量とスループット

ブロックファイルのダウンロード状況を調べるため、時間ごとの転送データ量とスループットを測定した。測定はサーバ側で tcpdump を用いた。latency がなしの場合を図 6,7、latency が 200msec の場合を図 8,9 に示した。

4.2.2.1 レイテンシがない場合

図 6 より分割圧縮ブロックファイル転送状況が判る。DLAHEAD がない場合では、起動処理でファイルアクセスがある度にダウンロードされていることが判る。10-20 秒の間で平坦になっているのは、KNOPPIX の Autoconfig によるデバイス認識を行っているため、ファイルアクセスがないからである。DLAHEAD があると先行ダウンロードを集中的に行うため、転送データ量はほぼ一直線に増加することが確認できる。DL-FULL では全て先行ダウンロードしてしまうので、終了後のダウンロードはない。DL-2500 では途中で先行ダウンロードが終了するため、残りのファイルアクセス要求があるまでダウンロードが停止する。アクセス要求が始まると DLAHEAD なしと同様にダウンロードされていることが確認できる。

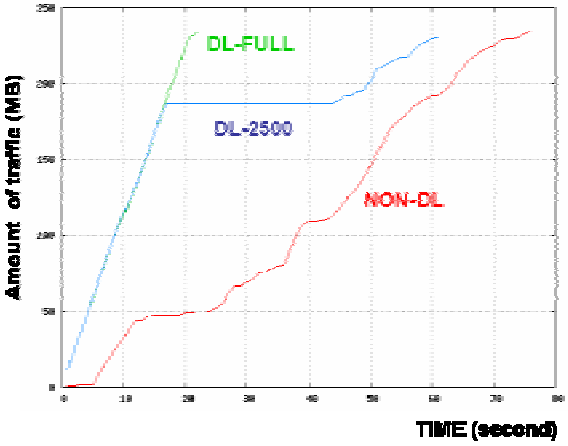

図 6 laency なしでの転送データ量

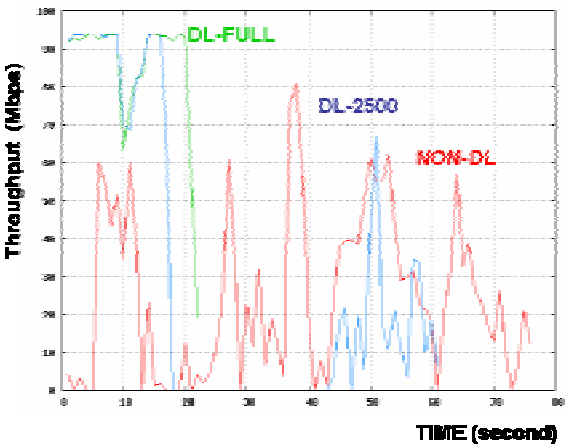

図 7 latency なしでのスループット

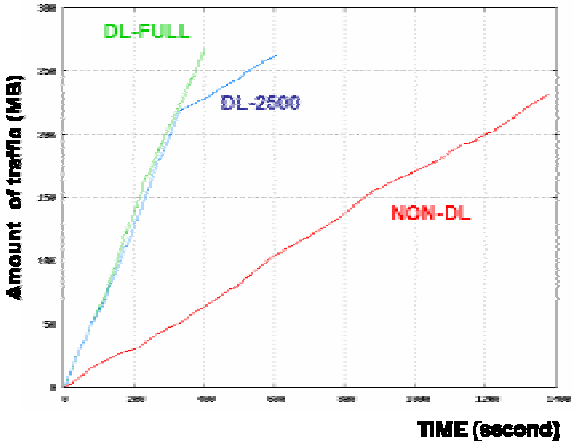

図 8 laency 200msec での転送データ量

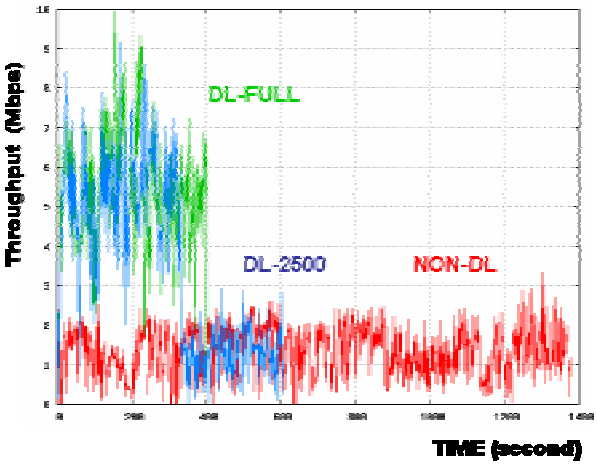

図 9 latency 200msec でのスループット

図 7 ではスループットの変化を示している。DLAHEAD があるとほぼネットワークの性能を使い切り、高スループット(95Mbps)を維持していることが判る。10 秒ぐらいの一時的な落ち込みは 4 本のダウンロードによるチョークと思われる。DLAHEAD なしの場合瞬間的な高スループットが起こるが、通常は低スループットで推移している。これは、HTTP-FUSE-CLOOP が仮想ブロックデバイスのためアクセス要求が逐次化され、細かいファイルをコネクション 1 本で読み出さなければいけないためである。また、表 3 のブロックファイル読み出しスループット(2.71MBps　21.7Mbps)と比べると HTTP-FUSE-CLOOP では高スループットが出ている。これは表 3 では wget を使ったのに対して、libcurl から呼び出していることが影響していると考えられる。

3.2.2.1 レイテンシが 200msec の場合

図 8 と図 6 を比べると、レイテンシの影響で転送が間延びしていることが判る。DLAHEAD では転送データ量はほぼ一直線に増加するが、その効率は格段に悪化する。また、ブートに必要な転送データ量は 234MB のはずであるが、DLAHEAD を適用した場合はいずれも 250MB を超える。これは DLAHREAD が HTTP-FUSE-CLOOP の読み出しに追い抜かれ、二重読み出しされているためである。現在の実装では二重読み出しを抑制する機構がないため、先行読み出しが追い抜かれたら余計なデータ転送を起こしてしまうが、抑制できるように改良する。

DLAHEAD なしの場合でもほぼ直線のダウンロードになっているが、これは読み出し要求に対して常にダウンロードネックになっているためである。200msec delay によるバンド幅縮小(Window サイズが 64KB の場合 2.5Mbps)がボトルネックなり、その性能が表面に出ている。

図８のスループットを見ると律速の状況が判る。DLAHEAD なしではほぼ 2.5Mbps 以下に抑えられている。DLAHEAD ありで４本の集中的なダウンロードを行っても 10Mbps 以下に抑えられ、効率的な転送ができていない。この状況は DLAHEAD のコネクション数を増やすことで改善されるが、元々のバンド幅が狭いので効果が薄い。できるだけ latency の短いサーバを見つけるのことの方が効果的であることが推測できる。

5. 議論：公開 HTTP サーバ

以上述べたような HTTP-FUSE KNOPPIX のソフトウェアを作成した。現在 KNOPPIX4.0 DVD 版対応のものを公開している。HTTP サーバ群としては ring プロジェクトのミラーサーバとニューヨーク大学の coral プロジェクトの P2P プロキシを利用している。

ring[14]は日本のフリーソフト配信の最大ミラーサーバであり、現在 20 以上のミラーが利用できる。地理的にもネットワーク的に分散さており、国内からの利用では netselect を用いて短いレイテンシのサーバを見つけることが可能となっている。

coral では[15,16]は .nyud.net:8090 を URL のサイト名に付けるのみで、P2P プロキシへ導かれる。例えば、www.aist.go.jp.nyud.net:8090/index.html により、www.aist.go.jp へのアクセスが P2P プロキシへ割り当てられる。これは .nyud.net:8090 の部分で名前解決を行う際にプロキシへの接続と導いているためである。coral は PlanetLab[17,18]を利用して、世界中に P2P プロキシを割り当てている。

coral は簡単に利用できるが、P2P プロキシが置かれているサイトが欧米に偏っており、国内からはその効果が確認できない。また、プロキシではファイルがキャッシュされていない場合、オリジナルサイトからのダウンロードを行わなければならず、その効率も確認しづらい。プロキシ同士でキャッシュを融通しあう ICP(Internet Cache Protocol)もあるが、あまり効率的ではないという報告[19]もある。このような環境ではプロキシサーバに対する計画なプッシュも必要と考えられる。

Akamai の FreeFlow によるエッジサーバサービスのようなものが簡単に利用できるとうれしいのだが現状では難しい。最近製品が出ている WAFS(Wide Area File System)は、WAFS アプライアンス機器を Internet と LAN の間に配置し、LAN 側では SMB/CIFS による通常のファイルシステムとしてのアクセス、Internet 側では別の WAFS アプライアンスと専用プロコルによる高速データ交換を実現している。WAFS アプライアンスがファイルのキャッシュと一貫性管理を行い、レイテンシ隠蔽を行っている。いずれにしろレイテンシを短く技術が重要であり、HTTP-FUSE-CLOOP においても同様のものが必要と考えている。

6.今後の課題

6.1 ファイル単位とブロック単位

HTTP-FUSE KNOPPIX ではデバイスブロックを分割して配信している。これに対して、WebDAV のようなファイル単位での配信も考えられる。しかし、ファイル単位では特殊ファイルが配信できないこと、ブロック差分のような透過的な更新が簡単にできないこと、大きなファイルだと無駄な転送が増えること、ファイルの詐称が簡単にできること、などの欠点がある。現在の実装ではこれらの欠点を克服できる仕組みがないためブロック単位の配信としたが、今後ファイル単位の配信手段についても考えていきたい。

6.2. セキュリティ

現在の版はハッシュ値(MD5)により内容確認でき

るようになっているが、セキュリティ的に充分でない。当面は内容確認できるためセキュリティを心配する必要がないと思うが、将来的には index.idx ファイルのみは https 配信すること、ブロックファイルの暗号化、ブロックファイルの署名、ダウンロードサーバの認証、などの対処を考えている。

7.おわりに

ブロックデバイスを分割圧縮したファイルから再構成できるループバックデバイス HTTP-FUSE-CLOOP を開発した。HTTP-FUSE-CLOOP は分割圧縮したファイルを手元の二次記憶からでも HTTP 経由のリモートからでも透過的に扱えるようにした。これを KNOPPIX に適用して HTTP-FUSE KNOPPIX を開発した。

分割圧縮ブロックファイルはコピーでも構わず、Proxy キャッシュなどをも活用できる。この利点はクライアント・サーバの束縛を離れ、広域に OS のイメージを配信可能とする。

HTTP-FUSE KONPPIX はとりあえず利用可能になったが、まだまだ一般に適用するにはレイテンシの問題やセキュリティの問題を詰める必要がある。今後はこれらの問題を解決し、使い勝手を高めていきたい。

参考文献&URL

[1] iSCSI, http://www.ietf.org/rfc/rfc3720.txt
[2] NFS4 http://www.nfsv4.org/
[3] Open-AFS, http://www.openafs.org/
[4] SFS, “http://www.fs.net/sfswww”
[5] SHFS, http://shfs.sourceforge.net/
[6] Quinlan, S. and Dorward, D.: Venti: a new approach to archival storage, the USENIX Conference on File and Storage Technologies, Monterey, CA, pp. 89–102 (2002).
[7] Dabek, F. Kaashoek, M. Karger, D. Morris R. Stoica,I.: “Wide-area cooperative storage with CFS”, 18th ACM Symposium on Operating Systems Principles (SOSP '01), (2001).
[8] Cohen, B.: “Incentives Build Robustness in BitTorrent”, First Workshop on Economics of Pexr-to-Peer Systems,(2003).
[9] Cox. B: BTSlave: http://btslave.sf.net/
[10] KNOPPIX, http://www.knopper.net/knoppix
[11] KNOPPIX 日本語版, http://unit.aist.go.jp/itri/knoppix/
[12]須崎,八木,飯島,丹, “HTTP-FUSE KNOPPIX”, ”,Linux Conference 2005.
http://lc.linux.or.jp/paper/lc2005/CP-02.pdf
[13] FUSE, “http://fuse.sourceforge.net/”
[14] Ring プロジェクト, http://www.ring.or.jp
[15] coral プロジェクト, http://www.coralcdn.org/
[16] M.J.Freedman, E.Freudenthal, and D.Mazières, “Democratizing Content Publication with Coral”, 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04)
http://www.coralcdn.org/docs/coral-nsdi04.pdf
[17] PlanetLab プロジェクト,http://www.planet-lab.org/
[18]L.Peterson and T.Roscoe “The Design Principles of PlanetLab”, Draft Paper, June 2004
http://www.planet-lab.org/PDN/PDN-04-021/pdn-04-021.pdf
[19]松本,河合,奥田,門, “Peer-to-Peer Network を用いた Web Cache の提案と実装”, DPS ワークショップ, (2002)