HTTP-FUSE KNOPPIX

http://unit.aist.go.jp/it/knoppix
須崎有康 1), 八木豊志樹 1), 飯島賢吾 1), 丹英之 2)
k.suzaki@aist.go.jp, yagi-toshiki@aist.go.jp, k-iijima@aist.go.jp, tanh@alpha.co.jp
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ファイルから再構成できるループバックデバイス FUSE-cloop を開発した。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ネットワーク経由のリモートからでも透過的に扱えるようにした。この分割圧縮したファイルを Internet の CDN(Contents Delivery Network)から HTTP 配信し、どのマシンでも起動できる 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。

HTTP-FUSE KNOPPIX

Kuniyasu Suzaki1), Toshiki Yagi1), Kengo Iijima1), Hideyuki Tan2)
National Institute of Advanced Industrial Science and Technology1),Alpha Systems Inc. 2)

Abstract ： We developed “FUSE-cloop” which can re-construct a virtual block device with split-and-compressed block files of block device. “FUSE-cloop” acts as transparent block device between local storage and network-download with split-and-compressed block files. Using FUSE-cloop, we made HTTP-FUSE KNOPPIX, which can get split-and-compressed block files from CDN(Contents Delivery Network) via HTTP.

1. はじめに

KNOPPIX[1,2]はブート時にデバイス自動認識/ドライバの組み込み機能(Autoconfig)が優れており、不特定多数のユーザが個々の PC からブートできる。また、ルートファイルシステムも圧縮ループバックデバイス(cloop: compressed loopback device)に格納され、１ファイルとして扱いやすい。しかし、現状では 700MB 近い cloop ファイルを全体として一括で扱わなければならない。このため、１つのアプリケーションでも変更した場合、cloop ファイル全体の変更となる。cloop が変更されると iso ファイルも作りなおさなければならず、700MB のダウンロード、CD の焼き直しの手間が発生する。

この問題を解決するために、cloop を拡張し、ファイルの読み出しがあった場合に Internet から必要な部分のみを取得する「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(FUSE-cloop)」を開発した。FUSE-cloop では仮想ファイルシステム FUSE[3]を利用した。この FUSE-cloop を使い、HTTP からルートファイルシステムを取得する KNOPPIX「HTTP-FUSE KNOPPIX」を開発した。

機能的には通常の CD 版 KNOPPIX と同じであるが、CD 版のように全てのルートファイルシステム(cloop ファイル)を持ち歩く必要はない。また、使い勝手は通常の CD 版と同じであり、ユーザはどこにルートファイルシステムがあるのか意識せずに利用可能となる。性能は CD ブートを標準として、それより早くなることを目指した。現実にはルートファイルシステムを Internet からオンデマンドのダウンロードをするため、ブート時間がネットワークの性能に依存するが、できるだけよいサーバを見つけ、ブート時間が短くなるような工夫をした。

我々は HTTP-FUSE KNOPPIX 開発以前に WAN 対応のファイルシステムである SFS[4](Self-certifying File System)経由で KNOPPIX のルートファイルシステム(cloop ファイル)を mount して使う SFS-KNOPPIX[5,6,7]を作成した。これによってクライアントとなる CD を作り直さなくても別のカスタマイズ KNOPPIX を使えるようになった。しかし SFS-KNOPPIX では単純なクライアント/サーバのためにレイテンシの長い海外からではブートに時間がかかる、ダウンロードした状態を保持できないためリブートした際に再 mount してダウンロードしなければならない、SFS 依存のため拡

張性がない、などの問題が表面化してきた。HTTP-FUSE KNOPPIX ではこれらの問題点を踏まえて改良した。

以下、2 章で FUSE-cloop の詳細、3 章で分割圧縮ブロックファイルの配信方法、4 章で現状の実装を報告する。5 章で関連研究と比較し差異を述べる。6 章で今後の課題、7 章でまとめを述べる。

2. 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 ファイルとして扱える。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り、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。

2.1 cloop の仕組み

cloop ではブロックデバイスを 64KB 毎に切り出し zlib 圧縮して、2G 程度のブロックデバイスを 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)には依存しない。

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。

```
# losetup /dev/cloop /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。下記のコマンド例では /dev/loop デバイスを /KNOPPIX に mount している。

```
# mount /dev/loop /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。KNOPPIX ではルートファイルシステム(/)が RAM-DISK に作られるので、/usr, /lib など必要なディレクトリは /KNOPPIX/usr, /KNOPPIX/lib から symbolic link が張られている。

この cloop 内のファイルアクセスは通常のファイルシステムと同じであるが、ブロックデバイスの読み出し手順が異なる。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、圧縮解凍を行って必要な部分のみを返す。

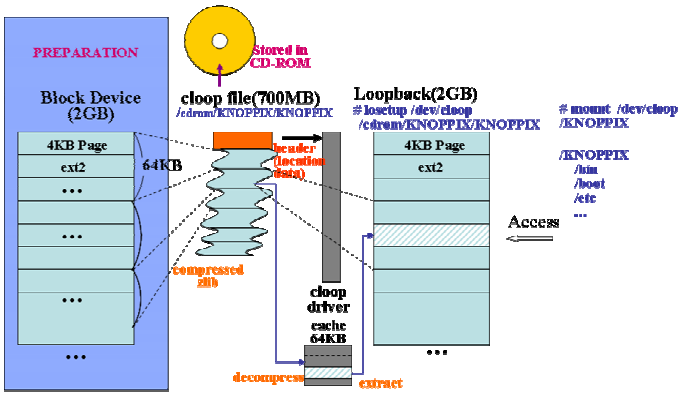

図 1 KNOPPIX での cloop の利用手順

2.2 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 700MB のファイルとして扱わなければいけないため、ダウンロードするに時間がかかる。

本開発では loop 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(FUSE-cloop)」を開発した。改良のポイントは下記である。

- 64KB 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)。
- 分割圧縮ブロックファイルをまとめて、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 checksum(Ex: md5)の値とし、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、checksum 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)[3]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 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 64KB 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が cloop として構成するために、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。

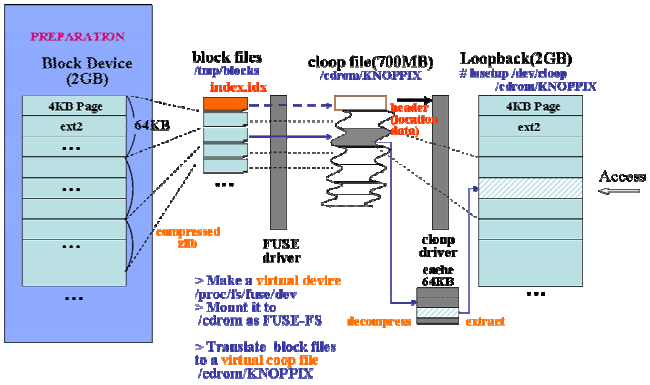

図 2 FUSE-cloop の構成

2.3. 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。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可能とした。

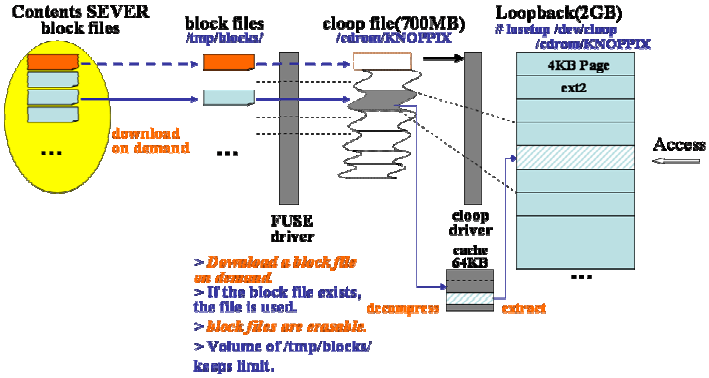

図 3 HTTP FUSE-cloop の構成

2.4. 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ようにした（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。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 のみの変更で差分更新に対応可能になった。

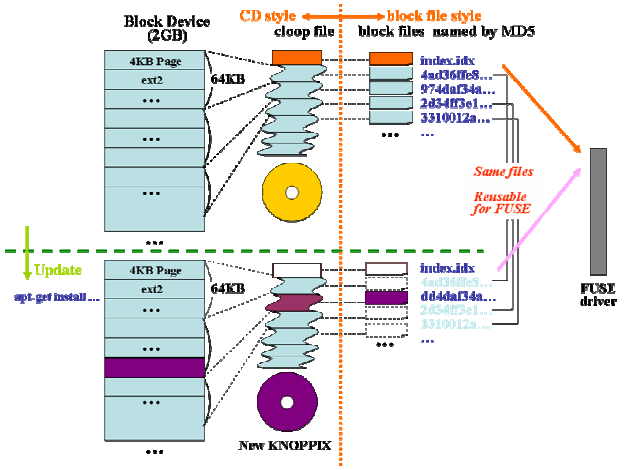

図 4 FUSE-cloop による差分更新

また、ファイル名は更新したブロックの内容の MD5 による新しいものとなる。FUSE-cloop では、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利用し続けることができる。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で同一 CD-ROM から別の KNOPPIX が利用できる。

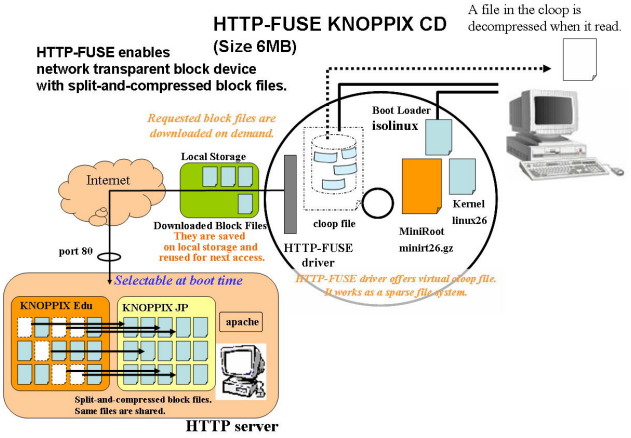

図 5 HTTP-FUSE KNOPPIX の構成

3. 配信手法

FUSE-cloop の作成により、分割圧縮ブロックファイルからブート可能になったが、Internet 経由でブートするにはダウンロードが高速でなければならない。また、安定性、信頼性を高めるためには単純なクライアント/サーバでは実現できない。その理由を以下で説明する。

3.1. ダウンロードする分割ブロックファイルのサイズ

「分割圧縮ブロックファイル」のダウンロード条件を詳細に調べると以下のようになった。

- (64KB 切り出しの場合)ブロックファイル
  - 31,206files(Max:65,562Min:84Ave:21,801)
- ブート時にダウンロードされるファイル
  - 4,424 files(87MB)
- ダウンロードパターン(ext2 へのアクセスパターン)
  - on demand (非連続)

分割ブロックファイルは大きさが平均 21KB と小さく、ダウンロードされるファイルの数が 4,000 以上となる。これらのファイルはオンデマンド（必要時に）要求され、基本的にアクセスパターンが予測でない。つまり、予めまとめてダウンロードすることができないことが判った。ここで要求される条件は、小さいファイルを素早くオンデマンドでダウンロードすることであった。

また、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 -> *(Ineternet)* -> file

cloop では、ブロックへのアクセス要求が逐次化される。これをネットワークに適応した場合、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（遅延）を受けないようにする手法であるが、FUSE-cloop ではその手法が適用できない。

また、ファイルサイズが 64KB での切り出しでは分割圧縮ブロックファイルのサイズが平均で 21KB となる。このサイズでは、ネットワークの Window サイズ 64KB（転送バッファ）に適さない。Window サイズの影響の詳細は図 6 のようになる[8]。

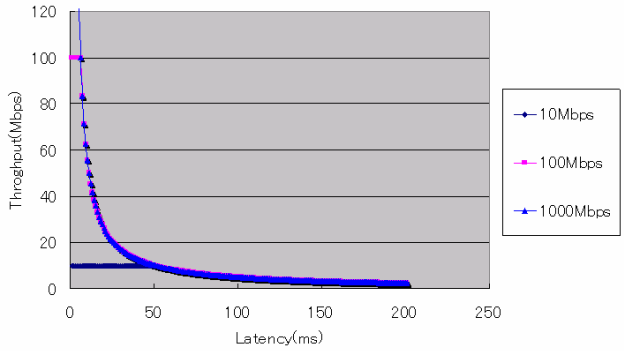

図 6 Window サイズ 64KB,コネクション 1 本のレイテンシとバンド幅の関係

(x がレイテンシ msec、y がバンド幅 Mbps)

x < 50 y=10, x >= 50, y= (50/x)*10　　(10Mbps の場合)
x < 5 y=100, x>=5, y=(5/x)*100　　(100Mbps の場合)
x< 0.5 y=1000, x>=0.5, y=(0.5/x)*1000 (1000Mbps の場合)

ここでは Window にデータを乗せる時間以上にレイテンシが長いとバンド幅が制限されることを示している。たとえば、100Mpbs では 64KB のデータを乗せるのに 64 * 1024 * 8 (bit) / 100,000,000 (bps) = 5.24 msec かかる。この時間以内に前の要求の ack が帰らないと送信が行えず、バンド幅に影響する。日米間では 100msec 以上のレイテンシが起こるため、100Mbps のネットワークを使っても実効バンド幅としては 5Mbps になってしまう。

64KB でブロックを切り出し、圧縮した圧縮ブロックファイルの転送では 64KB 以下のサイズになり、遅延が生じやすい。これを回避する一つの方法として、できるだけ転送データサイズを大きくしてレイテンシの影響を受けないようにする。幸いに FUSE-cloop では、切り出しブロックサイズを変更しても対応可能である。切り出しブロックサイズを大きくした場合の特性を表 1 にまとめた。

表 1　切り出しサイズの性質

	Files	Max (byte)	Min (byte)	Ave (byte)	download (files (sizeMB))	Full (%)
64KB	31,206	65,562	84	21,801	4,424 (87)	83
128KB	15,611	131,118	149	43,041	2,876(115)	72
256KB	7,821	262,230	277	85,372	2,089(171)	59
512KB	3,927	524,454	531	169,501	1,436(231)	45

#同一内容のファイルが存在する場合はファイル数が減る。
full はブートに実際使われる ext2 ページ割合

表 1 でファイル数が正確に半分にならないのは、同一内容のファイルは MD5 を取って 1 ファイルにまとめられるためである。また、Full の項目は実際にブートで必要となる ext2 ファイルシステムのページサイズの割合である。切り出しブロックを大きくした場合、含まれる必要ページ割合は減り、不要なダウンロードデータが増える。

更にブロックサイズを変更した場合の FUSE-cloop ドライバの性能を調べた。この性能評価では

dd if=/cdrom/KNOPPIX of=/dev/null

を実行して測定した。ここでは純粋にドライバの性能を調べるために RAM-DISK のブロックファイルを置いて測定した。また、ネットワークの影響も調べるため、簡単な HTTP ダウンロードを実装し、直結の HTTP サーバ(apache サーバ)からダウンロードした場合の測定も行った。その結果を下記にまとめる。

表 2 FUSE の性能

	Files	RAM-Disk (min:sec)	HTTP (min:sec)
64KB	31,206	9.96	1:58
128KB	15,611	10.59	1:42
256KB	7,821	20.91	1:43
512KB	3,927	54.34	2:12

#ThinkPADT42,PentiumM 1.8G,100MNIC,HTTP は直結
#レンテンシ(RTT< 0.2msec)

RAM-Disk の結果を見ると 256KB 以下の場合はドライバの影響は無視できそうである。ただし、512MKB の性能の劣化は CPU のキャッシュ性能を現しているようなので更に詳細を詰める必要があ

る。また、ここでの HTTP ダウンロードはレイテンシの影響を受けないため、切り出しブロックサイズが小さいほど効率的になるが、64KB では HTTP のダウンロード回数が影響して遅くなる。

更に HTTP 接続からブートする場合の時間を測定した。ブート時間は bootchart[9]を用いて行った。ここでのブート時間は init の終りから KDE の起動前までである。

表 2 bootchart による起動時間

	bootchart 時間 (ダウンロードファイル総サイズ、ファイル数)
64KB	0:43(87MB,4224)
128KB	0:57(115MB,2876)
256KB	1:01(171MB,2089)
512KB	1:16(231MB,1436)

#ThinkPADT42,PentiumM 1.8G,100MNIC,HTTP は直結
#レイテンシ(RTT< 0.2msec)

この結果では、切り出しブロックサイズが小さいほど性能がよいことが示されたが、レイテンシの影響を受けていないため実際の Internet 適用には更に影響は調べる必要がある。参考に CD ブートで同様の bootchart 時間は 2:15 となり、ネットワークブートでも CD より高速化できることが判った。

上記の測定より Window サイズの効率的利用と実際に利用されるページ割合などを考慮して、128KB あるいは、256KB のブロック切り出しが良いように思える。しかし、更に実際のレイテンシの影響などを含めて詳細を調べなければならない。現状では、256KB の切り出しブロックを用いて実装を行った。

3.2. ダウンロード方法の選択

ダウンロードロード手法として開発当初は P2P の Bittorrent を想定していたが、上記の調査により、Bittorrent には適さないことが判った。Bittorrent は大容量のファイルダウンロードを想定しており、ピースと呼ぶファイルに分割した後、複数のノードからピースを適当な順番でダウンロードして、全てダウンロードした後にファイルの再構成を行う。この技術は複数のダウンロードコネクションによるバンド幅拡張が基本的技術であり、オンデマンドに小さいファイルを要求する FUSE-cloop に適さない。

他のダウンロード手法も色々考慮したが、CDN(Contents Delivery Network)の技術を利用するのが最適だと判断した。CDN とは 1 つの HTTP URL で複数の配信方法を有するもので、ルーティング技術、キャッシュ、コンテンツ管理などの技術を有する仮想ネットワークである。具体的には、CDN 内に DNS が存在して、コンテンツ配信を行う場合にクライアントに適する (近い・負荷が少ない) サーバを割り当てるようにする。これを実現するために BGP を利用したレイテンシ短縮、サーバの負荷を計測してレスポンスの良いサーバへの割り当てなどの技術がある。サーバはミラーサーバ、プロキシサーバ（リバースプロキシ、P2P プロキシサーバ）などで複数のものが利用可能である。

利用対象とした CDN は下記のものである。

- ミラーサーバ群の ring プロジェクト[10]
  - 国内に 20 以上配置
- P2P proxy の coral プロジェクト[13,14]
  - PlanetLab を使い、世界中に配置。

3.2.1 ring プロジェクト

ring[10]は日本のフリーソフト配信の最大ミラーサーバである。国内の配置は下図に示すとおりである。

ring では tenbin[11], DNS balance[12]によるクライアントへ近いミラー選択、サーバの負荷分散の機能がある。

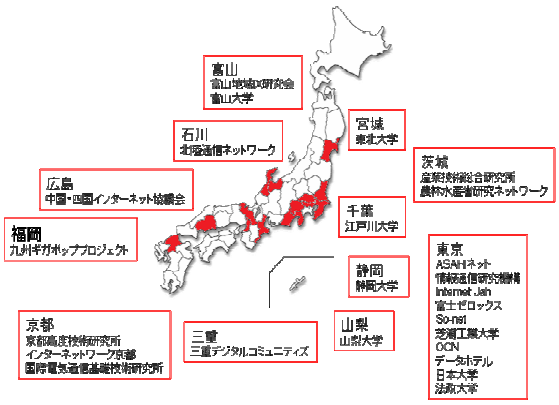

図 7 ring プロジェクトのミラー配置

3.2.2 coral プロジェクト

coral[13,14]はニューヨーク大学により研究されている P2P プロキシプロジェクトである。.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。たとえば、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と接続と導いているためである。また、coral では DSHT(Distributed Sloppy Hash Table)によりホットスポットを回避している。

coral は、PlanetLab[15,16]を利用して、世界中に P2P プロキシを割り当てている。その割り当て状況をした図 8 に示す。

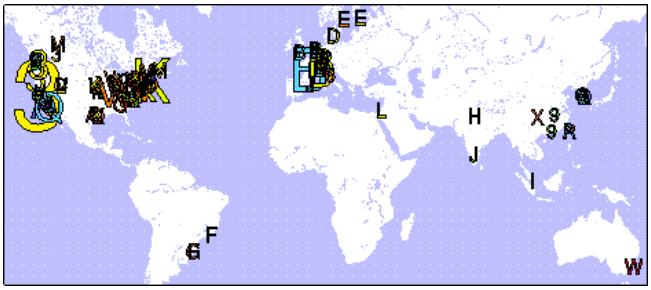

図 8 coral プロジェクトの proxy 配置(2005/04/03)

FUSE－cloop を使ったダウンロードではレイテンシの影響を受けやすいことがわかっている。特に欧米の接続は 100msec 以上のレイテンシがあることが判っており、できるだけ近くにサーバを配置する必要がある。coral は欧米に対して最適な proxy サーバの配置を行っているのでこれを活用する。

3.3. CDN とのネゴシエーション

FUSE-cloop を使った Internet からのブートでは、CDN のサーバに頼るだけでなくクライアント側でもレイテンシが短くなるように CDN を選択するようにした。つまり CDN とクライアントの相互のネゴシエーションによりダウンロード先が決定する。具体的には CDN が返す IP アドレス群を netselect をかけて選択する。図 9 がその様子を示す。

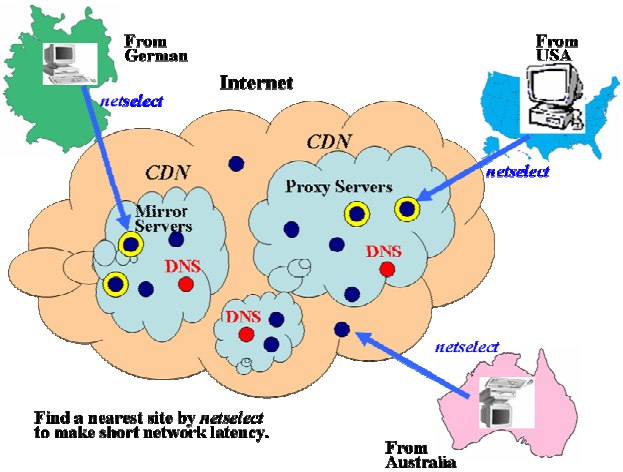

図 9 netselect でのサイト選択

netselect は traceroute によりクライアントから近いサーバを見つけるツールである。CDN の DNS が返す IP アドレスそれぞれに対して netselect を適用してサーバへの距離を計算する。そのうちで一番短いサーバから FUSE-cloop を通してダウンロードする。これより、高速なブートが期待できる。

4. 実装の現状

上記の機能を有するソフトウェアを作成し、KNOPPIX としてブートできるようにした。現在、USB メモリおよび CD-ROM からブートできるものを作成し、実行可能になっている。これらブートに必要なソフトウェアとしては 5MB 程度である。ブート後、ルートファイルシステム(FUSE-cloop)をマウントし、必要な分割圧縮ブロックファイルを Internet から取得する。

ダウンロードしたファイルは二次記憶に保存することも可能である。二次記憶のファイルシステムは Linux から書き込み可能であればよく、FAT でもかまわない。つまり、Linux 用に特別なパーティションやファイルシステムを用意する必要はない。2 度目以降のブートは二次記憶の分割圧縮ブロックファイルを利用できる。

そのブートの様子を示すと下記である。USB あるいは CD-ROM からのブート画面は図 10 になる。基本的にはオプションは不要である。

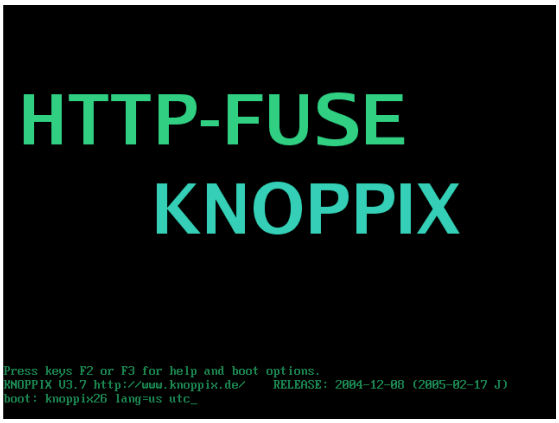

図 10 HTTP-FUSE KNOPPIX のブート画面

ブート後、miniroot の内容のみでネットワーク接続可能にする。cloop をマウントする前に分割圧縮ブロックファイルをダウンロードするサーバを選択する。その画面が図 11 である。netselect で選択する以外にブートオプションで個別のサイトに直接接続もできる。

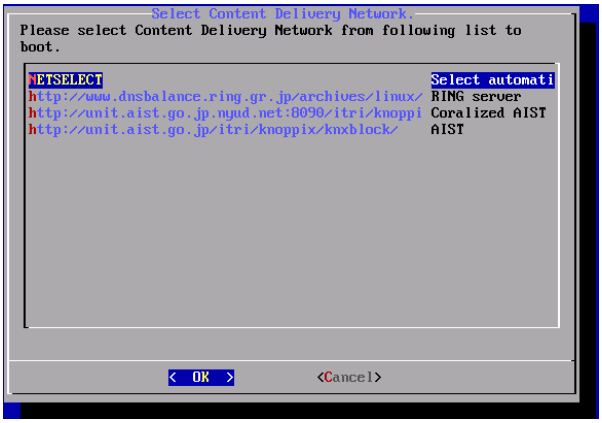

図 11 CDN 選択画面

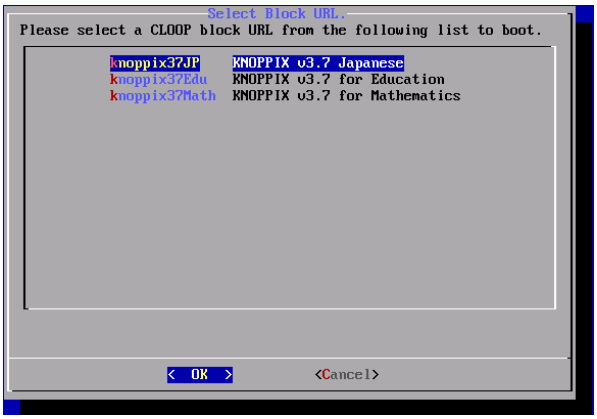

図 12 コンテンツ選択画面

次に KNOPPIX のコンテンツを選択する(図 12)。現在提供しているものは、KNOPPIX37JP(日本語版)とそれをベースにした Edu, Math である。それぞれ、256KB で切り出した分割圧縮ブロックファイルである。KNOPPIX37JP からの差分は Edu で 23%、Math で 62%であった。他の共通部分は KNOPPIX37JP と共有してサーバの容量を抑制できている。

図 13 netselect によるサイト選択

要求が決まれば、netselect が動作しクライアントから近いサイトを見つける。図 13 では netselect の値が最小の 2 となった aist.go.jp が選択されている。

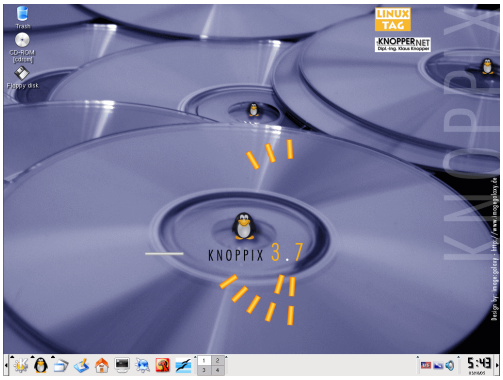

図 14 HTTP-FUSE KNOPPIX の起動

図 14 ではブート完了を表している。iptraf を実行すると、ルートファイルシステムにアクセスがある度にダウンロードが起こる様子が確認できる。

5. 関連研究との比較

5.1 Plan9 の Venti

Plan9 用に開発された archival block storage server の"Venti[17]"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基に行う。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 ハッシュによるストレージ管理を行っている点が似ているが、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は不要である。FUSE-cloop ではファイル配信さえ行えればよく、サーバ側に特殊なソフトウェアを想定しない。また、ファイル形式ならコピーも簡単に作成でき、proxy によるファイルコピーを使った Internet 配信を活用することができる。

5.2 locality-awareness を有する分散ファイルシステム

分散ファイルシステムである Coda[18], Shark[19,20]などには、サーバと切断しても cache の内容のみで利用可能な locality-awareness の性質を持つ。再接続した場合、更新ファイルの一貫性を保つ機能もある。

HTTP-FUSE でもダウンロードしたブロックのみを使う場合は locality-awareness の性質がある。ただし分散ファイルシステムと異なり、ブロックデバイスレベルの locality-awareness であり、該当ブロックの有る無しのみの機能しか提供しない。ファイルシステムの機能は HTTP-FUSE から見えないため、ディレクトリなどのメタデータのみのキャッシュなど機能分割ができない。また、cloop をベースにしているため、読み出し専用となる。機能的には locality-awareness を有する分散ファイルステムに劣るが、サーバがファイル配信の機能を有すればいいこと、また、ファイルのコピーを利用できることなど、サーバに依存しない構成になっている。

5.3 iSCSI

iSCSI は Internet からリモートの SCSI ブロックデバイスをアクセスできるプロトコルである。IETF において正式承認されている規格であり、ネットワークストレージの対応製品も多く出されている。

iSCSI は Internet 上でブロックデバイスレベルの扱いができる点は HTTP-FUSE と同じであるが、クライアント/サーバを基本にしており、HTTP-FUSE のようなサーバ非依存ではない。また、ブロックアクセスを Internet で飛ばすのみであり、locality-awareness の機能は無い。

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 ブート方法

現在の実装では最初のブートはまだ CD や USB など、BIOS からブートできるデバイスに依存する。しかし、この問題は Intel 社が中心に策定している次期 BIOS 規格 EFI (Extensible Firmware Interface)が広まれば OS のブートローダが BIOS に組み込めるようになり、直接ブートできる。これにより本方式の利便性を更に増すことを期待している。

6. 3 セキュリティ

現在の版は checksum 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。

6.4 レイテンシ

HTTP-FUSE KNOPPIX では世界中に分割圧縮ブロックファイルを配信するために P2P Proxy の coral 利用を想定している。しかし、coral は proxy のためキャッシュされていなければ proxy 自体がファイルをダウンロードしなければならない。幸い coral では coral 群内にファイルがキャッシュされていればそれを利用するようになっているが、その効率はよく判らない。確実にレイテンシを短くする配信については更に検討する必要がある。

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 FUSE-cloop を開発した。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、KNOPPIX に適用した HTTP-FUSE KNOPPIX を開発した。HTTP-FUSE KNOPPIX は先に開発した SFS-KNOPPIX と異なり、サーバに特殊なソフトウェアを仮定せず、ファイルさえ配信すれば利用可能になる。また、分割圧縮したブロックファイルはコピーでも構わず、Proxy Cache などを活用できる。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である。

HTTP-FUSE KNOPPIX では高速にブートする為にブロックファイルを配信するサイトのイテンシをできるだけ短くする必要がある。このため、複数の CDN と netselect でネゴシエーションする仕組みを加えた。現在、CDN としてミラーサーバ群の ring プロジェクトと Proxy サーバ群の coral プロジェクトに適用して利用可能にした。

また差分更新機能ではベースが KNOPPIX37JP (分割ブロックサイズ 256KB)とした場合、それからカスタマイズされた Edu、Math それぞれでは 23%、62%の更新差分を加えればよいことを確認した。

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。

謝辞

本研究の一部は情報処理推進機構(IPA)の平成 16 年度「未踏ソフトウェア創造事業」鵜飼文敏プロジェクトマネージャの「どこからともなくブートする OS」の成果である。共同開発者の岡島純氏(デジタルインフラ)との議論/発案により推進/実装されました。深謝します。

参考文献&URL

[1] KNOPPIX, http://www.knopper.net/knoppix
[2] KNOPPIX 日本語版, http://unit.aist.go.jp/itri/knoppix/
[3] FUSE, “http://fuse.sourceforge.net/”
[4] SFS, “http://www.fs.net/sfswww”
[5]須崎,飯島,丹,後藤, “SFS による UML-KNOPPIX サービスの性能評価”, ”,Linux Conference 2004. http://lc.linux.or.jp/paper/lc2004/CP-10.pdf
[6]後藤,須崎,飯島,八木,丹, “SFS を利用した KNOPPIX の起動と処理性能の評価”, 第 6 回インターネットテクノロジーワークショップ(WIT2004)
[7] K.Suzaki, K.Iijima, T.Yagi, H.Tan, and K.Goto, “SFS-KNOPPIX WHICH BOOTS FROM INTERNET”, International Conference on Web Information Systems and Technologies (WEBIST2005)
[8] A.Tanenbaum, Computer Networks, Prentice-Hall
[9] bootchart, http://www.bootchart.org
[10] Ring プロジェクト, http://www.ring.or.jp
[11] tenbin, http://t.ring.gr.jp
[12] DNS balance, http://www.dnsbalance.ring.gr.jp/
[13] coral プロジェクト, http://www.coralcdn.org/
[14] 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
[15] PlanetLab プロジェクト,http://www.planet-lab.org/
[16]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
[17]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).
[18] Coda, http://www.coda.cs.cmu.edu/
[19] Shark, http://www.scs.cs.nyu.edu/shark/
[20]S.Annapureddy, M.J.Freedman and D.Mazières,: “Shark: Scaling File Servers via Cooperative Caching”, 2st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '05)