HTTP プロトコルを用いる広域ストレージとそれを用いて起動する Linux

八木 豊志樹[1]　須崎 有康[1]　後藤 和弘[2]
飯島 賢吾[1]　丹 英之[3]

1. 独立行政法人 産業技術総合研究所
2. 大分県産業科学技術センター
3. 株式会社 アルファシステムズ

我々は、通信層に広く用いられているプロトコルである HTTP を使用する広域ストレージシステム HTTP-FUSE cloop および、HTTP-FUSE cloop を使って起動する Linux システム、HTTP-FUSE KNOPPIX の作成を行った。HTTP-FUSE cloop では、対象となるストレージをブロック単位で分割し、HTTP サーバ上にファイルとして配置するストレージサーバである。ブロックは、ストレージサーバから必要になったブロックを、HTTP を用いてダウンロードし、それを再構築し、仮想ストレージとして見せる。この方式により、ストレージサービスを気軽に扱うことができる。実装には、ユーザスペースにてファイルシステムの構築を行うことのできる FUSE、圧縮ループバックデバイスである cloop を用いた。cloop を用いているため、HTTP-FUSE cloop は読み出し専用となっている。本稿では、HTTP-FUSE cloop の設計方針およびその性能、今後の展望について述べる。

1. はじめに

我々は、インターネット上で OS の起動に必要なルートファイルシステムを公開し、そこから起動するクライアント OS の研究を行っている。不特定多数のユーザが、世界中のどこからでもインターネット接続ができていればハードディスクの必要がない OS となる。

インターネット上でルートファイルシステムを公開する手法としては、ブロックデバイスレベルの iSCSI[1]、ファイルシステムレベルの NFS4[2]、OpenAFS[3]、SFS[4]、SHFS[5]等が挙げられる。しかしながら、これらは複雑なプロトコルで、専用のポート番号やクライアントを必要とするため、ファイアウォールを越えた利用は難しくなっている。また、不特定多数のユーザが利用することを想定していない設計となっているため、われわれの目的とは相性がよくない。これらの問題を解決するため、われわれはルートファイルシステムを構成するストレージをブロック単位に分割し、それらをファイルとして HTTP サーバに配置し、クライアントが必要となったブロックをダウンロードし再構成を行う HTTP-FUSE cloop を開発した。

HTTP-FUSE cloop のクライアントは、サーバから対象ストレージのインデックスファイルを読み込み、その情報を元に仮想ブロックデバイスを作成する。そして仮想ブロックデバイスへアクセスがあった場合、対応するブロックファイルをオンデマンドでサーバからダウンロードし、内容を仮想ブロックデバイスに見せる。再度そのブロックにアクセスがあった場合、以前ダウンロードしたブロックファイルをローカルキャッシュから利用する。このため、必要なブロックファイルがキャッシュに残っていればネットワーク接続が切れていてもかまわない。また、サーバは既存の HTTP サーバであるため、負荷分散などは HTTP の手法を用いることができる。

HTTP-FUSE cloop のもうひとつの特徴は、ブロックへのアクセスに物理的なアドレスを指定するのではなく、ブロックの内容によるハッシュ値で検索する点である。これにより、重複する内容のブロックをひとつのブロックファイルにまとめることができ、またブロックに変更が起きた場合も変更前のブロックファイルを残しておくことによって、変更の巻き戻しを行うことも可能となる。このようなブロックレベルでの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方式である。

HTTP-FUSE cloop をルートファイルシステムとした HTTP-FUSE KNOPPIX を開発した。HTTP-FUSE KNOPPIX の起動に必要なファイルサイズは 5MB 程度であるが、これを用いることで通常の KNOPPIX と同様の機能を実現できる。KNOPPIX のファイルシステムに変更があった場

合、ダウンロードする HTTP-FUSE cloop を変更すればよいだけであり、CD を焼きなおす必要がない。

2. HTTP-FUSE cloop の概要

HTTP-FUSE cloop は、一つのストレージデバイスをブロックファイル単位で分割し、圧縮したものをファイルとして HTTP サーバに配置することによって、インターネットのどこでも利用することが出来る広域ストレージサービスである。この章では、HTTP-FUSE cloop の概要を説明する。

2.1 ブロックファイル分割

HTTP-FUSE cloop で使用することができるために、対象ストレージはブロックファイル単位で分割される。ブロックサイズは 512byte 単位で任意に決定することができるが、一つの仮想ストレージの中で異なるブロックサイズのブロックファイルを混在させることはできない。ブロックファイル分割プログラムを使用すると、対象ストレージはブロックサイズごとに分割され、圧縮されてファイルとして保存される。この時のブロックファイル名は、圧縮前のブロック内容の MD5 値となっている。すべてのブロックファイルが作成されると、対象ストレージの物理アドレスに対応したブロック位置のインデックスファイルが作成される。これを HTTP サーバにおくことによって、HTTP-FUSE cloop サービスを行うことができる。

2.2 HTTP-FUSE cloop の動作

HTTP-FUSE cloop を動作させると、サーバからインデックスファイルを読み込み、それを元に指定されたディレクトリに cloop ファイルを作成する。この時点では仮想ストレージである cloop ファイルの中は何も含まれない。仮想 cloop ファイルをループバックデバイスとして関連付け、そのデバイスにアクセスを行うと、HTTP-FUSE cloop にブロック読み出しリクエストが発行される。HTTP-FUSE cloop は、アクセスのあったブロックに対応するブロックファイルがローカルに存在すればそれをデバイスのブロックとして見せる。もしもブロックファイルがローカルに存在しなかった場合、オンデマンドでサーバにブロック読み込みリクエストを送出する。サーバからブロックファイルのダウンロードが終了したところで、ブロックファイルをローカルに保存し、それをデバイスのブロックとして見せる。

このため、既にブロックファイルがローカルにあれば、ネットワークの切断が起きてもサービス停止を起こすことがない。

2.3 ブロックファイルの差分更新

ストレージはブロックデバイスであるため、ストレージ更新が行われるとブロック単位でストレージ内容が変更される。HTTP-FUSE cloop ではブロックサイズ毎に分割してブロックファイルとして扱っているため、ストレージで変更のあったブロックに対応するブロックファイルのみが変更の影響を受ける。このため、HTTP-FUSE cloop においてストレージの更新を行う場合、変更のあったブロックファイルを追加することで実現できる。また、ブロックファイルに変更が起こると、識別子が変更前のものと異なるため、変更前のブロックファイルを消去する必要がない。

この性質を利用し、差分ブロックファイルとインデックスのみを更新することによってストレージの変更を行うことができる。また、変更前のファイルを保存しておくことによって、ストレージの履歴管理を行うことができる。

2.4 サーバ群の同期と接続負荷分散

サーバの負荷分散および冗長性の確保のために、サーバ間で同期を取り、接続するクライアントを分散させると便利である。サーバに存在するのはブロックファイルであるため、ブロックファイルのコピーを流通させることによってサーバ間の同期を取ることができる。具体的な手法としては、リバースプロキシを使う方法や、サーバ間を P2P などの方法で結合させる方法などが考えられる。

2.5 HTTP-FUSE cloop の実装

現段階の HTTP-FUSE cloop では、対象となるストレージを専用のコマンドを用いて分割を行っている。そのようにして作られたブロックファイルは、HTTP サーバに配置する、ローカルディスクに保存するなどの方法でクライアントに提供を行っている。ブロックファイルの識別子として用いているハッシュ値を求めるアルゴリズムに MD5 を用いている。

HTTP-FUSE cloop は、仮想ループバックファイルを提供するために Linux の FUSE(Filesystem on Userspace)[13]を用いている。FUSE のプログラムによって、デバイスのアドレス変換・ブロックファイルの読み込み・HTTP アクセスを行っている。

cloop を用いているため、デバイスへのアクセスは逐次化される。これが仮想ストレージへの並列アクセスをする際の障壁となっている。

3. HTTP-FUSE KNOPPIX

HTTP-FUSE cloop の上で動作するアプリケーションとして、HTTP-FUSE KNOPPIX[12]を作成した。これは、KNOPPIX[10][11]のルートファイルシステムを HTTP-FUSE cloop で置き換えたものである。HTTP-FUSE KNOPPIX を動作させるために必要なものは Linux カーネルと HTTP-FUSE KNOPPIX 用初期 RAM ディスクイメージで、その合計はおよそ 5MB 程度である。HTTP-FUSE KNOPPIX はルートファイルシステムを HTTP-FUSE cloop によりインターネット上に持ち、動作させるアプリケーションに必要なブロックファイルをユーザの選択した HTTP サーバからオンデマンドにダウンロードされる。通常の KNOPPIX では、ファイルシステム内のアプリケーションに変更があった場合、CD あるいは DVD を再作成する必要があるが、HTTP-FUSE KNOPPIX では HTTP サーバ上で差分更新を行うことで実現でき、CD を作り直す必要がない。

HTTP-FUSE cloop のための HTTP サーバとの距離が近く、十分な帯域が確保できれば、CD-ROM で KNOPPIX を起動するよりも高速に起動できる。また、HTTP-FUSE KNOPPIX のために HTTP-FUSE cloop にある特定のブロックファイルを先行読み込みする機能を付与した(dlahead)。これにより、HTTP-FUSE cloop へのアクセスが逐次化される問題をブート時に限り解消できる。

また、Linux カーネルと初期 RAM ディスクイメージを LAN から TFTP を使用してネットワークブートを行うことにより、完全にディスクレスの環境も作成した。これにより、HTTP-FUSE KNOPPIX のカーネル部に変更が起きても CD を作成しなおす必要がなくなった。

HTTP-FUSE KNOPPIX のブートプロセスを以下で説明する。

3.1 カーネルの読み込みとネットワーク設定

ブートのためには、カーネルと初期 RAM ディスクイメージを PC にロードさせる必要がある。HTTP-FUSE KNOPPIX では、カーネルと初期 RAM イメージの読み込みを行うために CD-ROM を用いる。HTTP-FUSE KNOPPIX CD-ROM を起動すると、図 1の画面が立ち上がる。boot: プロンプト以下でカーネルおよび HTTP-FUSE KNOPPIX に渡すオプションを指定できる。

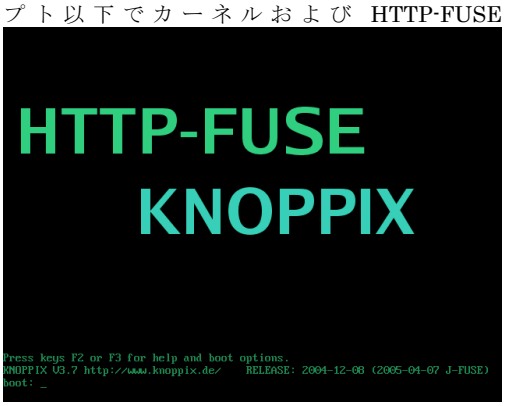

図 1 HTTP-FUSE KNOPPIX ブートメニュー

HTTP-FUSE KNOPPIX で指定できるオプションは以下の通りとなる。

- FUSE_URI={URI}
  ルートファイルシステムとする HTTP-FUSE cloop のベース URL を指定する。ここで file:/// を指定するとローカルファイルに存在する HTTP-FUSE cloop ブロックファイルを指定できる。
- MEMCACHE
  キャッシュを保存する場所に RAM ディスクを強制的に指定する。このオプションを指定しないと、HTTP-FUSE KNOPPIX は PC 上にある IDE/SCSI/USB デバイスをスキャンし、キャッシュを保存するのにふさわしいデバイスをマウントする。
- NOCACHE
  PC 上にキャッシュを保持しない。このオプションを指定すると、一度読み込まれたブロックであっても必要となるたびに HTTP へのアクセス要求が起こる。このオプションは、PC に搭載されているメモリが少ない場合に有効である。
- STATICIPADDRESS
  IP アドレスを静的に決定する。このオプションを指定すると、起動時にネットワーク設定(IP アドレス、ネットマスク、デフォルトゲートウェイ、ネームサーバ)を設定するプロンプ

トが出現する。

起動が始まると、まず USB、SCSI、ネットワークの各ドライバが読み込まれ、ネットワークの設定が行われ、RAM ディスク領域の作成を行う。

3.2 HTTP-FUSE cloop の設定およびマウント

FUSE_URI オプションを指定しなかった場合、図 2、図 3の画面が現れ、ルートファイルシステムとして用いる HTTP サーバおよびブロックファイルを選択できる。HTTP サーバ選択の際に netselect を選択すると、HTTP サーバリストにある各ホストとの距離を測り、ユーザに最適な HTTP サーバを選択できる。

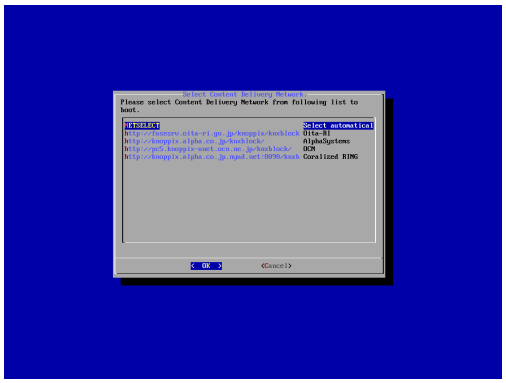

図 2 HTTP サーバの選択

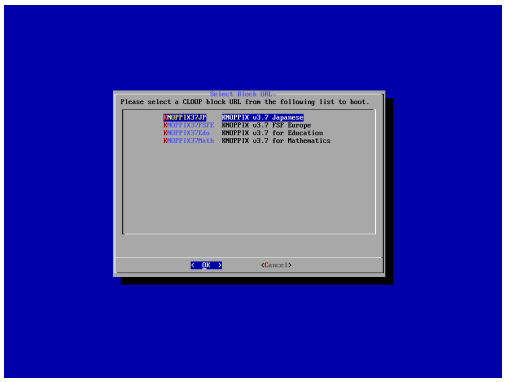

図 3 ルートファイルシステムの選択

ルートファイルシステムの選択が終了すると、図 4のように HTTP-FUSE cloop の設定およびルートファイルシステムとしてのマウントが行われる。ここから HTTP-FUSE cloop がインターネットへの接続を開始する。

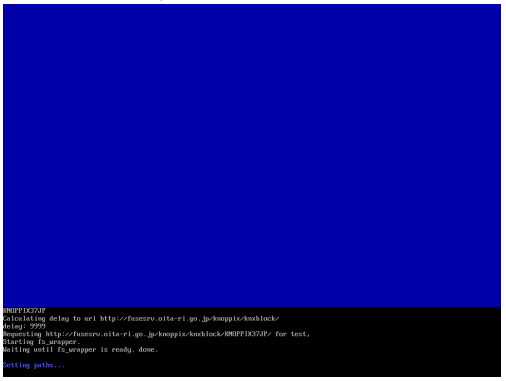

図 4 HTTP-FUSE cloop の起動

3.3 HTTP-FUSE KNOPPIX の起動

HTTP-FUSE cloop をマウントした後は、通常の KNOPPIX と同様に動作する。ここで、3.2にて選択したルートファイルシステムで起動するため、KNOPPIX の派生版(Edu/Math)なども、CD を作り直すことなく動作させることができる。

3.4 PXE ブートを用いた CD レス起動

通常の HTTP-FUSE KNOPPIX では、最初の起動の際に CD-ROM を使用しなければならなかったが、PXE ブート機構を用いると、PXE サーバにつながっているネットワークより HTTP-FUSE KNOPPIX を立ち上げることができる。現段階では、PXE を用いて HTTP-FUSE KNOPPIX を起動する際には PXE サーバ側にカーネルと初期 RAM ディスクイメージを配置する必要があるが、これは今後 HTTP 上にあるカーネルなどを、プロトコル変換を用いて完全にインターネット上に配置することを考えている。

4. 現在の改良点:更新可能な HTTP-FUSE loop

現在、ストレージへの書き込みを行うため、HTTP-FUSE cloop にてクライアント側に見せるループバックファイルを cloop から通常ループバックファイルに変更した HTTP-FUSE **loop** を試験的に開発した。これにより、cloop デバイスドライバの持つ「読み込み専用」という弱点を克服することができるようになった。また、HTTP-FUSE cloop の物と同一の圧縮分割ブロックファイルを用いることができる。

4.1 HTTP-FUSE loop の性能

今回作成した HTTP-FUSE **loop** と、既存の HTTP-FUSE cloop + cloop ドライバにて、どの程度の性能の違いが出るかを検証する。ターゲットとなる圧縮分割ブロックファイルを HTTP-FUSE loop および HTTP-FUSE cloop + cloop デバイスドライバを用いて各ループバックデバイスを全て読み出すのにかかった時間を計測する。読み出しは、GbE で繋いだ LAN 環境およびローカルの HDD を用いた。今回検証に用いたのは、KNOPPIX4.0 DVD の cloop ファイルを分割圧縮ブロックファイルにしたものである。展開後の容量は 5,767,430,144 バイトとなる。結果を表 1に示す。

	HTTP-FUSE **loop**	HTTP-FUSE cloop + cloop
GbE 経由	467.82sec(12MB/s)	262.37sec(22MB/s)
HDD 経由	315.32sec(18MB/s)	158.78sec(36MB/s)

表 1 HTTP-FUSE loop の読み込み性能

参考までに、同一サイズのファイルを読み込むのに要した時間は 106.79sec(54MB/s)であった。

HTTP-FUSE **loop** では、HTTP-FUSE cloop + cloop ドライバより 2 倍近く時間がかかっている。原因としては HTTP-FUSE **loop** にキャッシュ構造を持たせていない点であると考えられるため、実装の改良が必要である。

5. 今後の展望

HTTP-FUSE cloop を開発・動作検証をしている中で発見された問題点とその解消法、そして今後の機能拡張などを述べる。

5.1 ブロックアクセスの逐次化問題に対する解決策

性能測定を行っている最中、HTTP-FUSE cloop をファイルシステムとしてマウントした際に全てのアクセスが逐次化されている。これは、HTTP-FUSE cloop のマウント作業を行うために、一度 HTTP-FUSE cloop ファイルのループバックマウントを行っているためであることが原因である。これを解消するためには以下の方法が考えられる。

- HTTP-FUSE cloop を純然としたブロックデバイスとして作成する。これにより、ループバックマウントを行っていることに起因されるブロックアクセスの逐次化を解消することができる。
- HTTP-FUSE cloop の「HTTP 転送をブロック転送に使用する」コンセプトを用いたファイルシステムを作成する。これにより、アクセスの逐次化を解消できるのはもとより、よりネットワーク転送に適したブロック内ファイル配置を行うことが期待できる。

現在、HTTP-FUSE cloop をブロックデバイスとして作成する方向で調査を進めている。

5.2 ストレージへの書き込み

HTTP-FUSE cloop では、ブロックデバイスとして読み込み専用の cloop デバイスドライバを使用している。このため、HTTP-FUSE cloop は必然的に読み込み専用となってしまっている。この問題に関しては、4章であげた改良を行うことにより解消できる。また、ストレージへの書き込みを行った場合のローカルキャッシュ、サーバへの転送の問題もあるが、それに関しては調査を行っているところである。

5.3 ブロック内部のデータ配置

現在までの調査[14]により、圧縮分割ブロックファイルのブロックサイズを大きくしていくと、ブート時に必要な総ファイルサイズが増大していくことが判明している。これは、HTTP-FUSE KNOPPIX の上に ext2 ファイルシステムを搭載しているためであると考えられる。ext2 ファイルシステムは、読み書きを繰り返していくうちに発生するファイルの断片化の影響をより小さくするように設計されている。そのため、自動的にファイルの分散配置を行うため、HTTP-FUSE cloop のブロックの全てに必要なデータが格納されない問題点がある。

この問題を回避するために、HTTP-FUSE cloop の上に乗せるファイルシステムを iso9660 とする方法もある。しかし、iso9660 ファイルシステムは、データを順番に詰めていくため、ブロックファイルの差分更新ができない問題点がある。そのため、Ext2kai[15]などのファイルシステムに期待をしている。

5.4 ブロックファイルの配信とミラーリング

サーバとの遅延が大きくなるとバンド幅が小さくなってしまう。そのため、ユーザにより近いサーバを利用することができるようにする必要がある。

現在公開している HTTP-FUSE KNOPPIX v3.7 では、Ring プロジェクト[16]のミラーサーバとニ

ューヨーク大学の coral プロジェクト[17][18]の P2P プロキシを用いている。

Ring プロジェクトのミラーサーバによって、国内においては遅延の短いサーバを利用することができる。coral プロジェクトの P2P プロキシでは、サーバの末尾に".nyud.net:8090"を付与することにより、coral プロジェクト内にある、ユーザに最も近いプロキシに接続される。coral のサーバ群は Planetlab[19][20] を利用して、世界中に P2P プロキシを割り当てている。

coral の HTTP プロキシは HTTP/1.0 準拠のもので、HTTP/1.1 にて追加された Keep-Alive 機構が利用できないため、ブロックファイル毎に TCP の三段ハンドシェイクが必要となり、遅延が短い場合でもあまりバンド幅を稼ぐことができないことが判明している。

5.5 セキュリティ

HTTP-FUSE cloop では、ハッシュ値を求めるアルゴリズムとして MD5 を用いているが、MD5 の強衝突耐性を破る手法が実証されている[21]ため、より安全なアルゴリズムを採用する必要がある。

また、ブロックファイルについてはハッシュ値によりある程度の同一性を保証することはできるが、インデックスファイルに関してはそのような手法がとられていない。また、インデックスファイルに関しては同一性保証だけでなく、それが信頼できる機関から発行されているものであるかを確認する必要がある。これらを解決するためにインデックスファイルの署名および認証サーバなどの対処を考えている。

6. 終わりに

既存の手軽に扱えるプロトコルである HTTP を用いてインターネット上で使うことのできるストレージシステムである HTTP-FUSE cloop を作成した。また、そのアプリケーションとして、KNOPPIX のルートファイルシステムを HTTP-FUSE cloop にして小さい容量の CD で起動する HTTP-FUSE KNOPPIX を開発した。

HTTP-FUSE cloop では、ブロックファイルを単純コピーし、HTTP サーバに配置することによってミラーを簡単に作ることができる。また、プロキシなどを用いて、低遅延のアクセスを行うことを可能とする。

HTTP-FUSE cloop としての機能はひとまず利用可能であるが、デバイスへの多重アクセス、デバイスへの書き込み、セキュリティ面などで検討すべき点がある。今後はこれらを解決し、より使いやすいシステムを目指していく。

参 考 文 献

[1] iSCSI, http://www.ietf.org/rfc/rfc3720.txt

[2] NFS4 http://www.nfsv4.org/

[3] OpenAFS, 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 Peer-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] 八木, 須崎, 後藤, 飯島, 丹 "HTTP を用いた広域分散ストレージシステム" 第 4 回情報科学技術フォーラム L-001

[15] 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" 第 4 回情報科学技術フォーラム B-023

[16] Ring プロジェクト, http://www.ring.or.jp

[17] coral プロジェクト, http://www.coralcdn.org/

[18] 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

[19] PlanetLab プロジェクト, http://www.planet-lab.org/

[20] 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

[21] Xisoyan Wang and Hongbo Yu, "How to Break MD5 and Other Hash Functions", Eurocrypt 2005