L-001

HTTP を用いた広域分散ストレージシステム
Wide-area Distributed Storage System over HTTP

八木 豊志樹†、須崎 有康†、後藤 和弘‡、飯島 賢吾†、丹 英之††
Toshiki Yagi, Kuniyasu Suzaki, Kazuhiro Goto, Kengo Iijima, Hideyuki Tan

1． はじめに

近年、広域分散ファイルシステムやストレージシステムは、グリッドコンピューティングにおける重要な役割を担っている。しかしながら、Venti[1], Coda[2], Shark[3,4]などの既存の広域分散ファイルシステムでは専用のプロトコルを必要とするため、各拠点のファイアウォールによる通信の遮断が起きることが大きな障壁となっている。本論文では、広域分散ストレージシステムに用いるプロトコルとして HTTP を使用することにより、各拠点のファイアウォールに余計なルールを追加することなく実現できる手法を提案する。また、実際のアプリケーションとして、HTTP-FUSE KNOPPIX を紹介し、その性能等についての評価を行う。

2． 提案手法の概要

2．1． ブロックとサーバ・クライアント間通信

今回提案する手法では、1 つのストレージを一定のサイズのブロックに分割し、各ブロックを圧縮したものをブロックファイルとして保存し、ネットワーク上に配置する。サーバ・クライアント間の通信には HTTP を用い、クライアントはサーバから読み込んだブロックファイルを再構成し、ループバックデバイスとして 1 つのストレージに見せる(図 1)。クライアントは、必要なブロックファイルのみをオンデマンドでダウンロードし、またローカルキャッシュを持ち、ネットワークへの不要な負荷をかけないようにする。

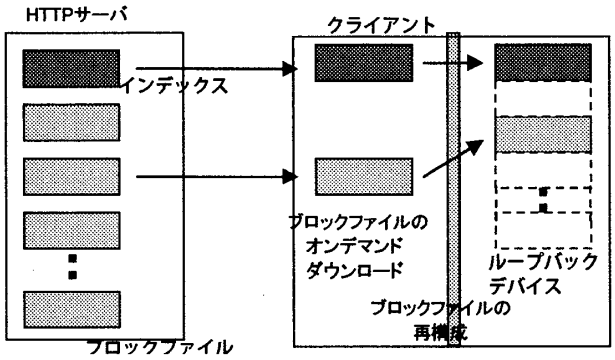

図 1 提案手法の概要

2．2． ブロックファイルの差分更新

ストレージを通常のファイルシステム経由で利用する際、更新が行われるとブロック単位でストレージの内容が変更される。このブロック単位でストレージを分割すると、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。この性質を利用し、差分ブロックファイル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。

2．3． 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。

3． 現在の実装と使用例

現在の実装では、1 つのストレージを 256kB に分割し、その MD5 ハッシュ値をファイル名としたブロックファイルを HTTP サーバ上に配置している。クライアント側は、Linux の FUSE(Filesystem on Userspace)[5]を用いてストレージを 1 つの cloop ファイルに見せ、それをループバックマウントすることによって読み込み専用のストレージとして扱っている。HTTP サーバ上に存在するブロックファイルは、cloop デバイスへのアクセスが発生するたびにオンデマンドでダウンロードされ、RAM ディスク等の外部記憶装置にキャッシュされる。

その上で動作するアプリケーションとして、HTTP-FUSE KNOPPIX[6]を作成した。カーネルと初期 RAM ディスクイメージの合計 4MB 程度で起動し、ルートファイルシステムは HTTP サーバ上からオンデマンドにダウンロードされ解釈される。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 で起動するよりも高速に起動できる。HTTP-FUSE KNOPPIX のために、また、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)。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。

4． 性能評価

4．1． 性能評価に用いたブロックファイル

現在の実装では、ブロックサイズを 512 バイトの整数倍を単位として自由に設定できるため、64kB、128kB、256kB、512kB の場合についての計測を行った。計測のターゲットは圧縮前の状態で 2GB の ext2 ファイルシステムのループバックファイルである。それぞれのブロックサイズで分割圧縮を行った結果を表 1に挙げる。

ファイル数と分割サイズを掛け合わせても分割前のファイルサイズとならないのは、ファイルを分割した結果同一のブロックファイルが作成されると同一ファイルとして扱

†(独)産業技術総合研究所
‡大分県産業科学技術センター
††(株)アルファシステムズ

われるためである。また、分割圧縮ファイルの Max 値は圧縮がまったく効かないブロックの場合、ヘッダサイズがその上に上乗せされるためである。

	#Files	Max(Byte)	Min(Byte)	Ave(Byte)
64kB	31,206	65,562	84	21,801
128kB	15,611	131,118	149	43,041
256kB	7,821	262,230	277	85,372
512kB	3,927	524,454	531	169,501

表 1 圧縮分割ファイルの性質

4．2．　デバイス全体の読み込み速度

ローカルの RAM ディスクにブロックファイルを置いた場合と、LAN 環境にある HTTP サーバにブロックファイルを置いた場合の性能を表 2に示す。ここでは、HTTP サーバに ThinkPad X30(Pentium iii 1.2GHz, 768MB, 100base NIC)、クライアントに ThinkPad T42(Pentium M 1.8GHz, 2GB, Gigabit NIC)を用い、この間をクロスケーブルで直結とした。

	64kB	128kB	256kB	512kB
RAM ディスク(sec)	9.96	10.6	20.9	54.3
LAN 上の HTTP(sec)	118	102	103	132

表 2　LAN 環境での性能評価

RAM ディスク上での読み込みは、ブロックサイズが大きくなるほど処理にかかる時間が増大した。これはブロックファイルのサイズが大きくなると、CPU のキャッシュに乗り切らなくなり、キャッシュミスが起こっているためと考えられる。LAN 上での HTTP からの読み込みについては、ブロックファイルのサイズが大きくなると HTTP リクエストの発行数が減少するため、ブロックサイズが大きくなると性能が向上していると考えられる。ただし、ブロックサイズが 512kB になるとローカルでの性能劣化が HTTP リクエスト現象の効果よりも大きくなってしまうため、全体として性能劣化が発生してしまっていると考えられる。

4．3．　HTTP-FUSE KNOPPIX の起動時間

4．2．での計測では、単純なストレージ読み出しにかかった時間を計測したが、ここではファイルシステムとして実際にこのシステムを用いた場合の性能を計測する。単純な読み出しとは異なり、ストレージへのアクセスパターンが離散的となる。KNOPPIX を起動し、実際に起動が完了するまでの時間を計測する。KNOPPIX の起動に必要となるファイルはデバイスドライバ読み込みの箇所を除き一定となるため、ブロックファイルの先行読み込みを行うこと(dlahead)ができる。128kB と 256kB の 2 つのブロックサイズに区切った同一内容のストレージにて、dlahead スレッドを 4 本走らせ、HTTP サーバとの遅延を変えながら計測を行った結果を表 3に示す。

	0ms	200ms	1000ms
128kB	66sec	181sec(489s)	855sec
256kB	74sec	205sec(585s)	980sec

表 3　HTTP-FUSE KNOPPIX 起動時間

この結果が示すとおり、ネットワーク遅延を付与すると読み込み性能は劇的に劣化する。また、ブロックファイル先行読み込み機能を無効にした場合、200msec の遅延を付与した場合の起動時間は 128kB 毎に分割した場合 489 秒、256kB 毎に分割した場合 585 秒となっていることから、先行読み込みの効果があることが確認された(200ms のカッコ内)。ブロックサイズが 128kB の方が 256kB よりも高速に起動したのは、ダウンロードされたブロックファイルのうち実際に使われたものの比(128kB:72%, 256kB: 59%)が原因になっていると考えられる。なお、オリジナル版の CD-ROM ブートにかかった時間は 135 秒であった。

5．　今後の展望

現在の実装では、リモートブロックファイルを再構成してストレージとして見せるために仮想ファイルシステムとループバックデバイスの 2 段構成を取っている。これをブロックデバイスドライバとして実装する方法を考案中である。また、現在の実装では、ストレージデバイスを cloop として表現しているため、ストレージデバイスへの書き込みを行うことができない。これをより汎用なデバイスとして表現することにより、読み書き可能なストレージデバイスを作成すること、それに伴うクライアントからサーバへのアップロード法なども検討する必要がある。

ブロックファイルやそれを管理するインデックスファイルなどは改ざんを受ける可能性があり、このような攻撃に対処するため、サーバからダウンロードしたファイルの正当性チェックを行う必要がある。ブロックファイルに関しては、ファイル名が内容の MD5 値と同一であるかのチェックを行えばその正当性を確認できる。ブロックファイルを管理するインデックスファイルに対して、どのように正当性チェックを行うかを検討中である。

6．　終わりに

既存のプロトコルを用いて広域分散ストレージシステムを構築する手法を提案した。これにより、現状のファイアウォールの制限内で簡単にストレージサーバを構築することができると考えられる。今後はストレージへの書き込みおよびストレージサーバへの書き戻し、変更のあったブロックファイルの分散方法を詰め、より利用しやすいシステムを作成することを目指す。

参考文献

[1] 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)

[2] Coda, http://www.coda.cs.cmu.edu/

[3] Shark, http://www.scs.cs.nyu.edu/shark/

[4] S. Annapureddy, M. J. Freedman and D. Mazieres, : “Shark: Scaling File Servers via Cooperative Caching”, 2nd USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI ’05)

[5] FUSE, http://fuse.sourceforge.net/

[6] HTTP-FUSE KNOPPIX, http://unit.aist.go.jp/itri/knoppix/http-fuse/