Vol.2011-OS-118 No.21
2011/7/28

重複除外によるGfarmの性能向上に関する検討

村上　じゅん[†1]　石黒　駿[†1]　大山恵弘[†1,†2]

Gfarmは，大規模データの効率的な利用を可能にする広域分散ファイルシステムである．本研究ではContent-Defined Chunking (CDC) による重複除外をGfarmに対して適用することを提案する．CDCはファイルをそのコンテンツに基づき可変長のブロック（チャンク）に分割する方式のことである．Gfarmのクライアントは，Gfarmファイルシステムから取得したファイルデータを，CDCを用いてチャンクに分割する．クライアントはローカルファイルシステム上にチャンクをキャッシュファイルとして保存し，以降のファイルアクセスで再利用する．重複除外の導入により，データ転送量やストレージ消費容量の削減が期待される．また，本研究ではCDCによる重複除外の効果を知るための予備実験を行った．

An Investigation of Improving Performance of Gfarm by Deduplication

Jun Murakami ,[†1] Shun Ishiguro [†1]
and Yoshihiro Oyama[†1,†2]

Gfarm is a global distributed file system that enables an efficient use of large-scale data. In this work, we propose an application of deduplication by Content-Defined Chunking (CDC) to Gfarm. CDC is a method of dividing a file into variable-size blocks (chunks) based on the contents of the file. File data obtained from the Gfarm file system is divided into chunks by the Gfarm client. The client stores the chunks in the local file system as cache files, and reuses them in the following file accesses. We expect that deduplication of chunks reduces the amount of transmitted data and storage consumption. We conducted a preliminary experiment for estimating the effects of the proposed deduplication.

†1 電気通信大学
The University of Electro-Communications
†2 独立行政法人科学技術振興機構，CREST
JST, CREST

1. はじめに

近年，大規模なデータの解析を必要とするアプリケーションのための広域分散ファイルシステムの研究が盛んに行われている．そのようなファイルシステムとして，Gfarm[1)] が挙げられる．Gfarmは大規模データ利用のために開発された広域分散ファイルシステムで，スケーラブルなファイルシステムの構築を可能にするものである．Gfarmの主要構成要素はファイルのメタデータを管理する単一のメタデータサーバ，ストレージを供給する複数のファイルシステムサーバ及びGfarmファイルシステムにアクセスするクライアントである．クライアントはGfarmが提供するライブラリAPIを呼び出すことによりGfarmファイルシステム上のデータを読み書きすることができる．

Gfarmにおいてはファイルアクセスは以下のように行われる．まずクライアントがGfarm上のファイルをアクセスすると，メタデータサーバとの通信の後，ファイルシステムサーバにファイルデータを要求し，ファイルデータを受け取る．そのデータは通常，クライアントのマシンに複製されて再利用されることはない．また，ページキャッシュに載ったとしても通常の設定ではファイルのオープンが実行されるたびにページキャッシュは捨てられる．これはデータの一貫性を保証するためである．よって，クライアントがGfarmファイルシステムにアクセスする際には，必ずファイルシステムサーバからファイルデータが転送されるようになっている．そこで一貫性を保証しながら，サーバから転送されたファイルデータをクライアントがローカルマシンに保存して再利用できるような仕組みが導入できれば，クライアントへのデータの読み込み性能が向上すると考えられる．

クライアントにキャッシュを作成する仕組みとしてまず考えられるのは，クライアントのローカルファイルシステム上にGfarmのファイルデータをファイル単位で保存しておくという方法である．この方法では，クライアントがファイルをオープンすると，そのファイルの複製がクライアントのマシン上に作成される．次にクライアントが同名のファイルをオープンした際には，ファイルシステムサーバからではなくクライアントの持っている複製からデータが読み込まれるという仕組みである．

しかし，この方法には問題点が2つある．1つはファイル全体をローカルファイルに保持するので，保持のために必要なストレージ容量がファイルサイズに等しく大きいことである．特に，ゲノムデータや物理実験データなどを扱うデータインテンシブサイエンスのアプ

リケーションにおいては，数百 GB などの非常に大きいファイルを利用することがある．

もう 1 つは，ファイルデータが更新される度に，保持しているローカルファイルも更新もしくは破棄しなければならず，ファイルの一部が頻繁に更新されるような場合においてはキャッシュの効果が発揮されないことである．

本研究では，Gfarm において，クライアントがファイルデータをその中身に基づいて可変長のブロック（チャンク）に分割してローカルファイルとして保存し，それらを後のアクセスで利用するキャッシュ機構を提案する．本研究で用いる分割方法は，一般に，Content-Defined Chunking (CDC）と呼ばれる．このキャッシュ機構には以下の利点がある．

- 冗長性がある複数のデータを 1 つにまとめる重複除外処理が実現できる．すなわち，データの中身が同一であるような複数のファイル断片に対して，同一のチャンクをキャッシュとして利用することが可能である．これにより，キャッシュによるローカルストレージの消費量を抑えつつデータ転送量を減少させることができる．
- ファイルが更新された場合におけるキャッシュの更新は，更新したデータを含むチャンクの更新によって行われるため，ファイル全体を更新する場合よりも更新するデータが少なくて済む．
- 過去にアクセスしたチャンクと中身が同一であるチャンクを含むファイルを転送する際にも，キャッシュが利用できる．

本論文の構成を以下に示す．2 章で Gfarm の概要，3 章で CDC を用いた重複除外処理について述べる．4 章で提案機構の設計と実装，5 章で CDC を用いた重複除外処理の効果を見積もるための予備実験についてそれぞれ述べる．最後に 6 章でまとめと今後の課題について述べる．

2. Gfarm

Gfarm は大規模データ利用のための広域分散ファイルシステムである．**図 1** に Gfarm のアーキテクチャの概要を示す．Gfarm の主な構成要素はファイルアクセスを行うクライアント，ファイルシステムサーバ，メタデータサーバの 3 つである．ファイルシステムサーバは Gfarm ファイルシステムにストレージを提供するサーバで，Gfarm ファイルシステム上に複数存在する．メタデータサーバはファイルのメタデータを管理するサーバで，Gfarm ファイルシステム上に一つだけ存在する．Gfarm ファイルシステムにおけるファイルは，ファイル単位またはファイル断片の形で複数のファイルシステムサーバ上に分散配置される．クライアントはファイルアクセスを行う際，まずメタデータサーバに問い合わせ，当該ファイルを格納するファイルシステムサーバについての情報を得る．そして得られたファイルシステムサーバの中から 1 つを選択し，ファイルデータを要求する． Gfarm ではファイルシステムへのアクセス手段として Gfarm 並列 I/O API を提供している．Gfarm 並列 I/O API はファイルのオープン，クローズといったアクセス手段を直接提供するインターフェースである．Gfarm 並列 I/O API の一部を**表 1** に示す．クライアントは gfs_pio_open を呼び出すことによって Gfarm ファイルシステム上のファイルをオープンすることができる．また，gfs_pio_read を呼び出すことによって，オープンしたファイルの内容を読むことができる．クライアントは Gfarm ファイルシステムを利用する際，直接または間接的にこれらの Gfarm 並列 I/O API を利用する．Gfarm のパッケージは，Gfarm 並列 I/O API のライブラリを提供している．クライアントはこのライブラリを自分のプログラムにリンクして呼び出すことにより，Gfarm ファイルシステムにアクセスすることができる．Gfarm ファイルシステムは，ユーザレベルのファイルシステムを構築するためのフレームワークである FUSE[2)] を用いることにより，Linux のファイルシステムにマウントできる．この場合にも間接的に Gfarm 並列 I/O API は呼び出される．

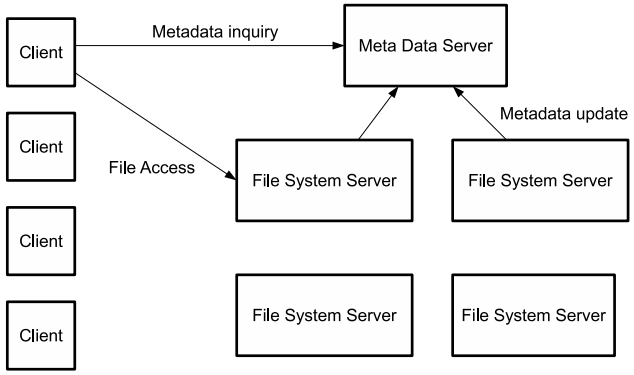

図 1 Gfarm のアーキテクチャ
Fig. 1 The architecture of Gfarm

よって，Gfarm 並列 I/O API に対し拡張機能を加えることで，あらゆるファイルアクセスに対応することができる． 本研究では，Gfarm 並列 I/O API のうちファイルの読み書

Vol.2011-OS-118 No.21
2011/7/28

表 1 Gfarm 並列 I/O API の一部
Table 1 A part of Gfarm parallel I/O API

gfs_pio_open	Gfarm ファイルのオープン
gfs_pio_read	ファイルの読み込み
gfs_pio_write	ファイルの書き込み
gfs_pio_close	Gfarm ファイルのクローズ

きに関するものに対し，本機構の導入を検討した．

3. CDC を用いた重複除外処理

3.1 CDC

CDC とは，ファイルの内容に基づいてファイルを可変長のチャンクに分割する方式である．LBFS[9] により用いられたのが最初であるが，その後多くのファイルシステムやストレージシステムに対して広く用いられた．重複したデータを持つチャンクをそのファイル内での位置によらず検出できることから，バックアップシステムの重複除外などでも用いられるようになった[8]．また，そのアルゴリズムやパラメータの設定等に関して数々の改良・最適化が提案されている[3][4]．

CDC ではチャンクの境界位置を定めるために固定長（典型的には 48 バイト長）のウインドウが用いられる．この方法では，ウインドウをファイルの先頭から 1 バイトずつスライドさせ，ウインドウに含まれるデータをハッシュ値に変換する．このハッシュ値の下位数ビットが特定の値と一致したときに，該当するウインドウの終点をチャンクの境界とみなすことで，ファイルの分割が行われる．CDC によるチャンク分割の例を**図 2** に示す．図中の左側の 2 つの濃い灰色の四角はチャンクの境界となる 48 バイトの領域である．右側の薄い灰色の四角は現在のウインドウを示す．四角から伸びる矢印の先にある値はその領域のデータから計算されるハッシュ値であり，図ではハッシュ値の下位 13 ビットが 0x1000 であるウインドウの終点をチャンクの境界としている． このときに下位何ビットまでを用いるかによってチャンクのサイズの平均が決まる．これは平均チャンクサイズと呼ばれる CDC のパラメータの一つである．また，このときのハッシュ関数として LBFS では Rabin fingerprint[5] が用いられている[9]．Rabin fingerprint はフィンガープリントの一種で，数学的には 2 元体上の既約多項式の除算に基づいたアルゴリズムである．与えられた長さ n のビット列を $n-1$ 次の多項式とみなし，これをある決められた k 次既約多項式で割る．このときの余りは $k-1$ 次の多項式であり，長さ k のビット列に対応するので，これを Rabin fingerprint

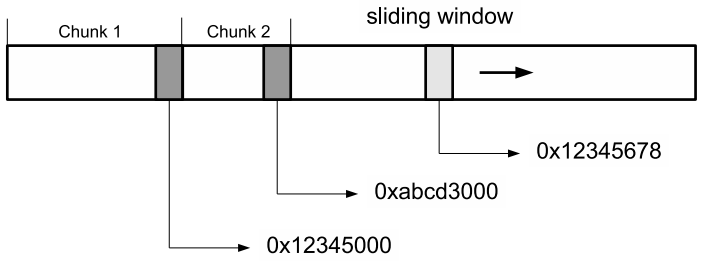

図 2 CDC によるチャンク分割
Fig. 2 Chunking by a CDC method

とするものである．Rabin fingerprint は効率的な計算が可能であるという特徴を持つ．より具体的には，バイト列 $a_0...a_{n-1}$ から求めたハッシュ値を利用してバイト列 $a_1...a_n$ の計算が高速に行えるという性質を持つ．チャンクへの分割においても，1 つ前のウィンドウのハッシュ値を利用して，現在のウィンドウのハッシュ値を効率的に求めることができる．さらに予め計算結果を格納したテーブルを用いた高速化手法も提案されており[6]，本研究でもこの手法を利用した．

CDC は主なパラメータとしてウインドウサイズ，平均チャンクサイズを持つが，その中でも性能に直接大きな影響を及ぼすのは平均チャンクサイズである．平均チャンクサイズが小さいほどチャンクの重複率は向上する[11] が，同時にチャンクを管理するためのオーバーヘッドも増加することが予想される．適切な平均チャンクサイズはアプリケーションやデータに依存するが，LBFS の例では平均チャンクサイズは 8KB に設定されている[9]．本研究でもこの値を採用した．

3.2 重複除外における CDC の利用

次に，CDC を用いた重複除外処理の方法[7][8] について説明する．

CDC をストレージの重複除外に適用する際，ファイルはファイル単位でなくチャンク単位で保存される．また，チャンクはそのチャンクの中身により識別される．具体的にはチャンクの中身を元にそのハッシュ値を計算し，その値をチャンク ID として用いるといった方法が用いられる．これにより，中身の等しいチャンクは等しいハッシュ値，すなわち等しいチャンク ID を持つことになる．ここで，中身の等しいチャンクはそのストレージ内にただ一つしか作成しないことにすると，重複するチャンクは作成されないことになり，その結果

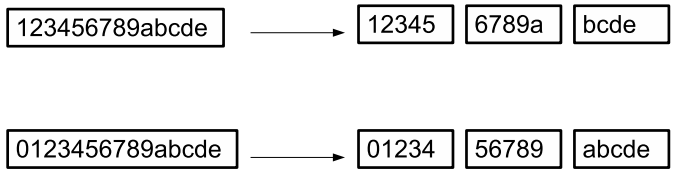

図 3 1 バイト付加したファイルのブロック分割（ブロックサイズ 5bytes）
Fig. 3 Blocks of a file which added one byte (block size is 5 bytes)

として重複除外処理が行われる．

CDC の利点は，ファイルの一部が変更された際に，そのファイルのチャンクへの変更が少なくて済むことである．ファイルを固定長のブロックで分割した場合，ファイルの先頭や途中に対して，データの追加や削除が行われたときに，以降の全てのブロックがずれてしまう問題がある．**図 3** にファイルの先頭に 1 バイトのデータが追加された場合の例を示す．左上の四角はデータの追加前のファイルを表しており，矢印の先の複数の四角はファイルの先頭から 5 バイト長のブロックに分割した結果を表す．そのファイルの先頭に 1 バイトのデータを追加したものが左下の四角であり，データ追加後のファイルをブロック分割した結果がその右の四角である．図より，データの追加前後で全てのブロックの中身が変化しているのがわかる．このように，ファイルの先頭や途中に対してデータの追加や削除が行われると，ファイルが以前とは別の中身を持つブロックに分割されてしまい，重複除外が働かなくなってしまう．

一方，CDC による可変長チャンクを利用した場合は，影響を受けるのは最初のチャンクのみであり，二番目以降のチャンクは変化しない可能性が高い．また，ファイルの途中にデータを挿入した際も同様に，挿入箇所を含むチャンクにしかほとんど影響を及ぼさない．というのも，チャンクの境界はファイルデータで決まるため，ファイルデータのうち変更のない部分はチャンクの境界も変化しないからである．そのため，類似データをチャンクに分割した結果が一致しやすくなり，重複除外が比較的有効に働くと考えられる．実際，文献 11) の研究における実験では，CDC は固定長ブロックによる分割と比較してファイルの冗長性の検出性能が高いという結果が得られている．

4. 提 案 機 構

4.1 設　　計

提案する機構は，基本的には LBFS[9] で用いられている手法に従っている．本機構は Gfarm 上のファイルの 1 回目の read，write 時に，クライアントのローカルディスクにキャッシュを作成する．キャッシュの作成では，前章で述べた CDC による重複除外処理を適用し，チャンクごとにキャッシュファイルを作成する．

同一内容のキャッシュファイルはただひとつしか作成されないとする．これにより，チャンク単位の重複除外が実現する．キャッシュヒットの判定は，チャンク ID の一致・不一致により行う．これにより，初めて読み込むファイルであっても以前読み込んだファイルと同じチャンクを持っていればキャッシュの効果が期待できる．また，ファイルが更新されてもキャッシュを捨てる必要はない．なぜなら，ファイルが更新された時点で更新部分を含むチャンクが変化し，それによりチャンク ID も変化するため，次のファイル read 時にキャッシュが読まれることはないためである．なお，チャンク ID を求めるためのハッシュ関数としては，LBFS などの例に倣い SHA-1 ハッシュ[12] を用いた．

本機構導入後のファイル読み出し処理の流れは以下のようになる．

(1) クライアントがメタデータサーバにファイルのパスを送信し，そのファイルの内容を持っているファイルシステムサーバの情報を得る
(2) クライアントがファイルシステムサーバに対してファイルオープンを発行する
(3) ファイルシステムサーバはファイルを CDC によりチャンク分割し，全チャンクのハッシュ値およびファイルオフセットをクライアントに送る
(4) クライアントは read 時に読み出し部分を含むチャンクのハッシュ値を用いてそのチャンクがキャッシュされているかどうかを調べる
(5) もし同一のハッシュ値を持つチャンクが存在する場合はそのチャンクのデータを読み込む
(6) 同一ハッシュ値を持つチャンクが存在しない場合はファイルシステムサーバからデータを読み込む
(7) ファイルシステムサーバから送られたデータは CDC によりチャンク分割され，新たなチャンクファイルが生成される

図 4 にファイルアクセスの概要を示す．ファイルシステムサーバがクライアントに返すハッシュ値は，図のようにファイルオフセットと合わせて送られる．

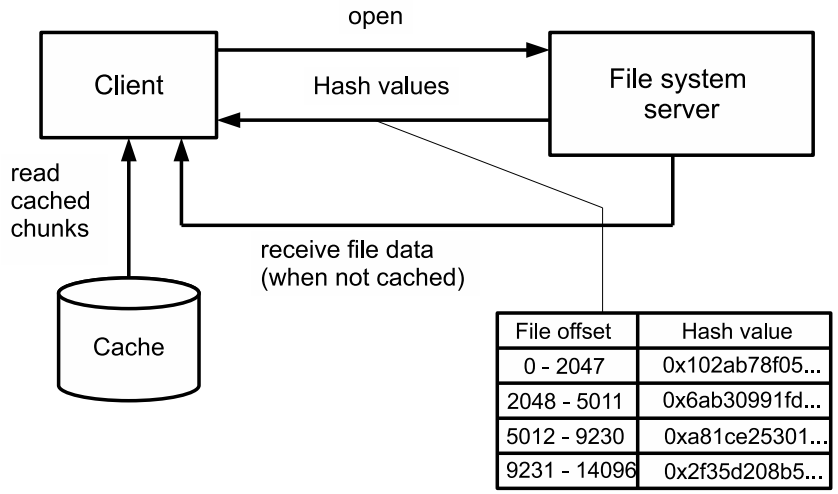

図 4 提案機構導入後のファイルアクセス
Fig. 4 File access using our mechanism

ファイルシステムサーバから送られたデータのチャンク分割であるが，送られたデータは必ずしもファイルの先頭から始まるとは限らないので，データの先頭はチャンク境界であるとは限らない．例えばクライアントがファイルの途中に seek して読もうとした場合，サーバから送られるのはファイル途中からのデータである．このような場合を考慮し，本機構では送られたデータにより生成されたチャンクデータのうちファイルの先頭および末尾のデータを含むチャンクはキャッシュを生成せずに捨てるとした．

クライアントの側だけではなく，ファイルシステムサーバの側でも，一度作られたチャンクファイルは保存され，以降のファイルアクセスで再利用される．また，ファイルシステムサーバは，ファイルをチャンクに分割する際に，全チャンクのハッシュ値とファイルオフセットを記録したファイルを作成する．クライアントからのファイルオープンでそのファイルは利用される．

4.2 実　　装

CDC により与えられたファイルをチャンク分割し，キャッシュファイルを生成するプログラムの実装を行った．キャッシュファイルはチャンク ID をファイル名として持ち，チャンクデータを中身として持つようなファイルである．

表 2 実験環境
Table 2 experimental environment

CPU	Intel Core 2 Duo 3.16GHz
OS	Linux Debian lenny
Kernel	2.6.26
Memory	2GB

表 3 アプリケーションが読み込むファイルの重複除外
Table 3 Deduplication of files read by applications

	平均チャンクサイズ	重複率
Firefox	8KB	0%
	1KB	0.3%
	64B	2.2%
カーネルコンパイル	8KB	0%
	1KB	0.1%
	64B	4.6%

Gfarm ファイルシステムへの組み込み部分は現在，実装中である．

5. 予 備 実 験

今回は予備実験として，CDC を用いたチャンク分割をファイル群に適用し，適用前後のデータ量を比較した．ファイル群は複数のファイルパスの集合である．今回の実験では Firefox の起動時および Linux カーネルのコンパイル時に読み込まれるファイル群に対して，それぞれ CDC による重複除外を適用した．実験には Firefox 5.0, Linux kernel 2.6.39.1, gcc 4.3.2 を使用した．また，実験環境は**表 2** の通りである．結果は**表 3** のようになった．なお，表 3 の重複率は全ファイルのデータサイズから全チャンクのデータサイズを引いたもの（重複データ）を全ファイルのデータサイズで割ることにより算出している．

平均チャンクサイズを 8KB に設定した場合は Firefox, カーネルコンパイルのいずれも重複するチャンクが生成されず，重複除外の効果は得られなかった．一方，平均チャンクサイズが 64 バイトの場合は重複したチャンクがファイルデータ全体に対してそれぞれ 2.2%, 4.6% 生成された．この実験では重複除外の効果が必ずしも大きくはなかった．その理由として考えられるのは，Firefox もカーネルのコンパイルも，必ずしも巨大なファイルや冗長性が高いファイルを扱うわけではないことである．今後，科学技術計算などで用いられる，巨大で冗長性の高いファイルを利用するアプリケーションを用いて実験を行い，提案機構に

より実現できる重複除外の効果をさらに調査していく予定である．

6. 関 連 研 究

LBFS[9] は低バンド幅ネットワークのためのファイルシステムであり，CDC を初めて提案したシステムであるとされている．Pastiche[8] は CDC を用いたバックアップシステムである．Shark[10] は cooperative cache を実現するファイルシステムである．クライアントのキャッシュに対し CDC による重複除外を適用し，さらにクライアント同士がお互いにキャッシュを利用し合うことで並列処理を行い，バンド幅を高めている．

以上のように，CDC を用いた重複除外は様々なシステムで適用され，その効果が実験等により確認されている．これらを踏まえ，本研究では Gfarm に対し CDC による重複除外の適用を提案した．

7. お わ り に

本論文では，Gfarm ファイルシステムの性能向上を目的とした重複除外処理システムの適用を提案した．また予備実験により，提案システムを導入した際の重複除外効果を調べた．今後は，Gfarm に本システムを組み込んだ場合の性能について調べる予定である．

謝　　辞

本研究を進めるにあたり，有益な助言をして下さった筑波大学建部研究室の方々に心より感謝する．また本研究は，科学技術振興機構戦略的創造研究推進事業（JST CREST）の研究課題「ポストペタスケールデータインテンシブサイエンスのためのシステムソフトウェア」の支援を受けている．

参 考 文 献

1) Osamu Tatebe, Kohei Hiraga and Noriyuki Soda : Gfarm Grid File System, New Generation Computing, Ohmsha, Ltd. and Springer, Vol. 28, No. 3, pp.257–275 (2010).
2) Szeredi, M : FUSE: Filesystem in Userspace, http://fuse.sourceforge.net/.
3) E. Kruus, C. Ungureanu, and C. Dubnicki : Bimodal Content Defined Chunking for Backup Streams, In Proceedings of the 8th USENIX Conference on File and Storage Technologies, pp.239–252 (2010).
4) Jaehong Min, Daeyoung Yoon and Youjip Won : Efficient Deduplication Techniques for Modern Backup Operation, IEEE Transactions on Computers, Volume 60, Issue 6, pp.824–840 (2010).
5) Michael O. Rabin : Fingerprinting by random polynomials, Center for Research in Computing Technology, Harvard University, Technical Report TR-15–81 (1981).
6) A. Z. Broder : Some applications of Rabin's fingerprinting method, Sequences II: Methods in Communications Security and Computer Science, pp.143–152 (1993).
7) Tim D. Moreton, Ian A. Pratt and Timothy L. Harris : Storage, Mutability and Naming in Pasta, In Proceedings of Networking 2002 Workshops, Volume 2376 of Lecture Notes in Computer Science, pp.215–219 (2002).
8) Landon P. Cox, Christopher D. Murray and Brian D.Noble : Pastiche: Making Backup Cheap and Easy, In Proceedings of the 5th Symposium on Operating Systems Design and Implementation, pp.285–298 (2002).
9) Athicha Muthitacharoen, Benjie Chen and David Mazières : A Low-bandwidth Network File System, In Proceedings of the 18th ACM Symposium on Operating Systems Principles, pp.174–187 (2001).
10) Siddhartha Annapureddy, Michael J. Freedman and David Mazières : Shark : Scaling File Servers via Cooperative Caching, In Proceedings of the 2nd Conference on Symposium on Networked Systems Design and Implementation, pp.129–142 (2005).
11) Calicrates Policroniades and Ian Pratt : Alternatives for Detecting Redundancy in Storage Systems Data, In Proceedings of the USENIX 2004 Annual Technical Conference, pp.73-86 (2004).
12) J. Burrows : Secure Hash Standard, Federal Information Processing Standards Publication (1995).