ネットワーク上のマシンをディスクキャッシュに利用した場合の性能評価

上田　高徳[†]　　平手　勇宇[††]　　山名　早人[†††]

† 早稲田大学理工学部 〒169-8555 東京都新宿区大久保 3-4-1
†† 早稲田大学大学院理工学研究科 〒169-8555 東京都新宿区大久保 3-4-1
††† 早稲田大学理工学術院 〒169-8555 東京都新宿区大久保 3-4-1
国立情報学研究所 〒101-8430 東京都千代田区一ツ橋 2-1-2
E-mail: †{ueda,hirate,yamana}@yama.info.waseda.ac.jp

あらまし　ディスクアクセスを可能な限り回避するため，OS はディスクキャッシュ機能を備えている．しかし，空きメモリ容量が少ない場合，ディスクキャッシュ領域を充分に確保できない．この場合ディスクアクセスが頻繁に発生し，システムの性能が低下する．この問題を解決するために，本論文ではネットワークに接続されたリモートマシンの物理メモリを利用することでローカルディスクキャッシュの領域を拡張する手法を提案する．これまで，リモートマシンのメモリ資源を有効に利用する研究が行われてきた．しかし，リモートマシンのメモリをローカルディスクキャッシュに利用する手法は確立されていない．本論文においては Linux 2.6.15 のカーネルを修正することで提案手法を実現した．そして，修正した Linux 上において DBT-3 により PostgreSQL に対するベンチマークを行ったところ，最大 3.08 倍の性能向上が確認できた．

キーワード　ディスクキャッシュ，ベンチマーク，性能評価，データベース

Performance Evaluation of using Machines on a Network as Disk Cache

Takanori UEDA[†], Yu HIRATE[††], and Hayato YAMANA[†††]

† Faculty of Science and Engineering, Waseda University 3-4-1 Okubo, Shinjuku-ku, Tokyo 169-8555 Japan
†† Graduate School of Science and Engineering, Waseda University 3-4-1 Okubo, Shinjuku-ku, Tokyo 169-8555 Japan
††† Science and Engineering, Waseda University 3-4-1 Okubo, Shinjuku-ku, Tokyo 169-8555 Japan
National Institute of Informatics 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430 Japan
E-mail: †{ueda,hirate,yamana}@yama.info.waseda.ac.jp

Abstract　Modern operating systems have disk cache mechanism to reduce frequency of disk accesses. But when computers do not have enough physical memory or computers run many applications which allocate much memory, operating systems are not able to allocate enough local disk cache area, since they use free physical memory for local disk cache. This situation causes many disk accesses and goes down system performance. To solve this problem, we propose a system for increasing the amount of local disk cache by using physical memory of machines connected to a network. We have implemented our system on Linux Kernel 2.6.15, and benchmarked PostgreSQL by DBT-3 on the kernel. The PostgreSQL benchmark results show that our proposal system makes computers' faster up to 3.08 times.

Key words　Disk Cache, Benchmark, Performance Evaluation, Database

1. はじめに

ハードディスク (以下，単にディスク) は構造上の理由から，半導体メモリと比較してアクセス速度が低速である．特にランダムアクセス時はヘッドのシーク動作が必要となるため，著しくアクセス速度が低下する．これらの理由から，ディスクアクセスをチューニングすることでシステムの性能向上を実現できる．しかし，ディスクアクセスのチューニング手法は，ハードウェアとアプリケーションの組み合わせによって異なる．個々の環境に合わせてチューニングを行うことは，最適なチューニングが可能である反面，労力と費用を要する．より簡便に既存のシステムにおいて効果が得られる手法が望ましい．

そこで本研究では，OS が備える機能であるディスクキャッシュに注目した．リスト 1 はファイルからデータを読み込む際のプログラム例である．

```
int fd = open("name", O_RDONLY);
char buf[4096];
int ret = read(fd, buf, sizeof(buf));
```

リスト 1　read システムコール

ここで，read は POSIX 準拠の OS システムコールであり，char 型の配列 buf に最大 sizeof(buf)=4096 バイト読み込むように指示している．read の返り値は実際に読み込まれたデータサイズ (byte) である．

通常，OS は read システムコールにより読み込まれたデータを空きメモリ内にキャッシュしているため，再度同じファイルを読み込む時にキャッシュから高速に読み込むことが可能である．このディスクキャッシュの効果により，特別なチューニングをせずともファイルにアクセスするアプリケーションの高速化が期待できる．しかし，OS はディスクキャッシュの領域に空きメモリ領域を利用するため，メモリの搭載量が少ないマシンの場合や，メモリを大量に利用するアプリケーションが動作している場合，ディスクキャッシュ領域を充分に確保できない．この場合，ディスクアクセスが発生し，ディスクキャッシュにヒットした場合よりもシステム性能が低下する．

そこで本論文では，ネットワークに接続されたリモートマシンの物理メモリを用いてディスクキャッシュ領域を拡張する手法を提案する．ネットワークとして現在広く利用されている Gigabit Ethernet の帯域は，CPU とメモリ間の帯域はもちろん，ローカルハードディスクのインターフェース帯域よりも劣る．しかし，リモートマシンのメモリは半導体メモリであるため，ランダムアクセス時はローカルディスクに対してアクセスするよりもリモートマシンのメモリにアクセスした方が高速という特徴がある．この特徴を生かして，ランダムアクセスされるファイルをリモートマシンのメモリにキャッシュすることで，ディスクアクセスの高速化を図ることが可能である．

本論文は，ローカルディスクのキャッシュの拡張としてリモートマシンのメモリを用いる．これまでも，リモートマシンのメモリ資源を有効に利用する研究はなされてきた．しかし，リモートマシンのメモリをローカルディスクキャッシュに利用する手法は実用化されていない．本論文は，実用化を目指して実装を行ったプロトタイプの報告である．本論文では Linux カーネル 2.6.15 を修正することで提案手法を実現した．

本論文の構成は次の通りである．2. では関連研究について述べ，3. では提案手法について詳細を述べる．そして 4. で提案手法を実装した Linux 上でベンチマークを行った結果を述べ，5. でまとめを行う．

2. 関連研究と関連製品

これまで，ネットワークに接続されたリモートマシンのメモリを有効に利用する研究が行われてきた [1]〜[5]．リモートマシンのメモリの利用モデルを提案した Remote Memory [1] を筆頭に，ネットワーク経由で物理メモリ上のデータを共有する分散共有メモリ [5] といった研究が 1990 年代を中心に行われた．また近年では，クラスタ環境においてスワップ先に他のノードを利用するといった研究 [3] も行われている．

リモートマシンのメモリを利用する際の問題はネットワークの帯域である．ネットワーク経由でのリモートマシンのメモリへのアクセスは，ローカル物理メモリにアクセスするよりも遅い．そのため，ローカル物理メモリの代替としてリモートマシンのメモリを利用する場合，性能低下をいかに抑えるかが研究の主要課題となる．

これに対して，ローカルディスクのインターフェースはローカルメモリバスほど高速ではなく，リモートマシンのメモリをローカルディスクの代替に利用することで，場合によってはシステムの高速化を図ることができる．例えば，ディスクはランダムアクセスにおいて低速であるため，Memory Servers [2] の実験のように，リモートマシンのメモリをスワップに利用することでスワップ発生時のシステム高速化が可能である．また Nswap [3] のように，クラスタ環境において他ノードの空き物理メモリをスワップ領域に利用すれば，スワップ発生時のクラスタシステム高速化が可能である．

これらスワップ高速化の既存研究は，リモートマシンのメモリをディスクの代替として利用することで性能向上を図れることを示している．ただし，スワップが発生するのはメモリ不足という緊急的場面であり，スワップの高速化は直ちにシステムの性能向上には繋がらない．これに対して本論文の手法では，スワップ領域ではなく，ディスクキャッシュ領域を拡張するためにリモートマシンのメモリを利用する．ディスクキャッシュ領域の拡張であれば，スワップ発生時という限られた状況でなくとも，ファイルにアクセスするアプリケーションの高速化が期待できる．

ディスクキャッシュ領域をリモートマシンのメモリではなく，PC に付加されたデバイス上にとり高速化する製品は既に存在している．具体的には，Microsoft [6] が提唱する ReadyBoost [7] と ReadyDrive [7]，そして Intel [8] が提唱する Robson [9] といった製品である．

ReadyBoost は Windows Vista から新しく搭載された機能であり，USB メモリをディスクキャッシュ拡張に利用する．普及している USB メモリを簡単に利用できる点が長所である．突然 USB メモリが外されても，データの損失がないように配慮されている．また，セキュリティ対策のため，USB メモリに書き込まれるデータは暗号化される．

ReadyDrive はディスクドライブ本体にフラッシュメモリを搭載し，ディスクとフラッシュメモリをハイブリッドに使用する手法である．ReadyDrive を利用するには対応ディスクドライブを用いる必要があるため，すでに動作しているシステムに適用するにはドライブの交換が必要となる．また，OS と連携してフラッシュメモリの利用を制御するため，Windows Vista における利用が前提となっている．

Intel が提唱する Robson は，PCI Express スロットにハードディスク専用のキャッシュカードを装着する方式である．2007

年以降に発売予定の Intel のチップセットを搭載したマザーボードで利用できる．PCI Express の帯域は高速であるが，Robson で利用されるのは不揮発性メモリであり，動作速度は搭載される不揮発性メモリの速度により制限される．

こうした従来研究・現行製品がある中で，本論文では，リモートマシンのメモリをローカルディスクのキャッシュとして利用した場合の性能評価をするべく実装を行った．これまでも，たとえば [2] において，ローカルディスクのキャッシュとしてリモートマシンのメモリを利用できるという記述はあったものの，実現はされていない．本論文は，現在の PC 環境での実用化の可能性を探るべく作成したプロトタイプの報告である．過去のリモートマシンのメモリを利用する研究 [1]〜[5] は，研究時期の関係で 100Mbps 以下のネットワーク環境において行われており，本論文の意義は 1Gbps におけるデータを取得した点にもあると考える．環境の変更を余儀なくされる現行製品に対して，本論文の実装では，既存環境の Linux カーネルを入れ替えれば導入ができるという利点がある．

3. 提 案 手 法

3.1 概　　要

図 1 に提案手法の概念図を示す．

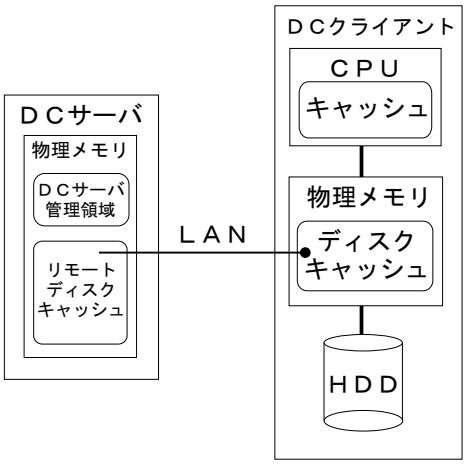

図 1　提案手法の概念図

本提案手法はネットワーク上のマシンの物理メモリをディスクキャッシュとして利用する．以下，ディスクキャッシュとして物理メモリを提供するネットワーク上のマシンを DC(Disk Cache) サーバと略す．また，DC サーバをディスクキャッシュとして利用するマシンを DC クライアントと略す．

ネットワークに接続されたリモートマシンの物理メモリをディスクキャッシュに利用する概念自体は，ハードウェアや OS に制限されるものではない．しかし，本論文では提案手法を実装する OS に Linux を選択しており，本論文の手法は Linux に依存する部分がある．

3.2 提案手法の特徴

以下に提案手法の特徴をまとめる．以下の節でこれらを説明する．

- Linux2.6.15 が動作していれば，カーネルの置き換えで導入が可能．
- DC サーバはユーザプログラムとして実装可能．
- 過去の PC 資産を DC サーバとして再利用が可能．
- ネットワーク接続が失われてもデータ損失無し．
- Linux の先読み機構と協調したキャッシュ動作．
- Direct I/O に対しては機能しない．

3.3 必 要 環 境

提案手法の実装は Linux カーネル 2.6.15 に行った．DC クライアントへの導入は Linux カーネルの入れ替えのみで良く，既存のアプリケーションは再コンパイル・再インストールの必要がない．DC サーバのプログラムはユーザプログラムとして実装可能なため，DC クライアントと同じネットワークに接続できれば DC サーバの OS 種別は問わない．すなわち，DC サーバをディスクレスで稼動させることも可能である．また，DC サーバに過去の PC 資産を活用することも可能である．

3.4 帯域の問題

本手法で最大の問題となるのが帯域である．表 1 は各種インターフェース速度を示したものである．

表 1　各種インターフェース速度

USB 1.1 (Full Speed)	12Mbps (1.5MB/sec)
USB 2.0 (High Speed)	480Mbps (60MB/sec)
1000Base-T	1Gbps (125MB/sec)
Serial ATA 1.0	150MB/sec
Ultra160 SCSI	160MB/sec
Ultra320 SCSI	320MB/sec

1000Base-T の理論値は 1Gbps (　125Mbytes/sec) であるが，実効転送速度は 1Gbps よりも遅い．このため，DC サーバからデータを読み込むよりも，直接ローカルディスクから読み込んだ方が高速な場合がある．しかし，本提案手法においてディスクキャッシュとして利用するリモートマシンのメモリは半導体メモリである．半導体メモリはランダムアクセスでも高速という特徴がある．そのため，ランダムアクセスされるファイルを優先的に DC サーバにキャッシュするといったキャッシュアルゴリズムの実装により性能の向上を実現できる．

3.5 耐障害性とセキュリティ

本手法はネットワーク経由でデータを転送するため，耐障害性とセキュリティが問題となる．耐障害性を確保するため，本論文における実装では，write back キャッシュとはしていない．すなわち，DC サーバ上のキャッシュデータは，DC クライアントのディスク上か，DC クライアントのディスクキャッシュ上に必ず存在する．そのため，障害により DC サーバと DC クライアント間の接続が失われたとしてもデータ損失は発生しない．write back キャッシュとすることで，ランダム書き込みの性能を向上できる可能性がある．write back キャッシュの検討は今後の課題とする．

セキュリティ対策のためには，ReadyBoost のようにキャッシュデータの暗号化を行う必要がある．しかし，現在の実装では暗号化に対応していないため，DC サーバ用にプライベートネットワークを構築して運用することが望ましい．通信の暗号化も今後の検討課題とする．

3.6 Linux カーネル

本節では Linux カーネルの機能のうち，本論文に関係する部分について概要を説明する．ここではカーネルバージョン 2.6.15 に基づいて説明を行う．

3.6.1 システムコール

リスト 1 で示した通り，ファイルを読む際には open でファイルを開いたあと read を利用する．ファイルに書き込む際は write を利用する．これら open，read/write はシステムコールである．システムコールは，呼び出したプロセスの延長上で動作する．そのため，シーケンシャルリードを行うプロセスが複数並行して動作していた場合，ディスクにおいてはランダムアクセスが発生する．このようなケースは今後到来するマルチコア時代において多く発生すると考えられ，より一層のディスク高速化技術が求められる．本提案手法は，このようなマルチタスクに起因するランダムアクセスの高速化にも対応している．

3.6.2 ページキャッシュ

Linux カーネルにおいて物理メモリ上のディスクキャッシュはページキャッシュと呼ばれる．read の場合，ページキャッシュにデータがあれば，ページキャッシュからユーザ空間にデータがコピーされる．ページキャッシュにデータが無ければ，ディスクからページキャッシュにデータを読み込んだあと，ページキャッシュからユーザ空間にコピーされる．

write の場合，すぐにはディスクへの反映は行わず，まずページキャッシュ上にデータが保持される．ディスクへの反映は一定時間後か，アプリケーションからリクエストがあった場合，もしくは，ディスクに未反映のページキャッシュ容量が閾値よりも多くなった時に行われる．

通常，ページキャッシュには空き物理メモリが最大限に利用され，空きメモリが不足した際に開放される．開放の際，write されたページキャッシュのうちディスクへの反映が必要なものは，ディスクに反映が行われる．

これらページキャッシュの機能により，同じファイルが繰り返しアクセスされる場合，ユーザ空間へ高速にデータを転送できる．一方で，カーネルによるページキャッシュ管理のオーバーヘッドが入るという欠点もある．例えば read でディスクから読み込まれる場合，ディスクからページキャッシュへ，そしてページキャッシュからユーザ空間へ転送するため，最悪 2 回のコピーが発生する．この余分なコピーはファイルを 1 度しか読まない場合には明らかに無駄である．また，大容量のファイルを読み込むとページキャッシュが溢れ，それまでキャッシュされていたデータが破棄されてしまうという問題もある．これらの問題を回避するため，Direct I/O と呼ばれるページキャッシュを経由しないアクセス手段が用意されている．Direct I/O を用いれば，ディスク上のファイルにページキャッシュを介さずアクセスすることが可能になる．Direct I/O を用いることで，アプリケーションが自身に適したキャッシュ機構を実装し，ディスクアクセスを高度にチューニングすることが可能となる．

本論文における実装では，Direct I/O によるアクセスに関してはキャッシュを行わない機構とした．そのため，Direct I/O を行うアプリケーションに対しては高速化の効果が得られない．Direct I/O 時も強制的に本手法を機能させる実装を行うことは不可能ではないが，オーバーヘッドを挿入することになり，性能を低下させる可能性がある．

3.7 提案手法におけるキャッシュ管理

Linux は read/write のリクエストがあったファイルがページキャッシュ上にあるかをソフトウェアで管理している．同様に提案手法でも，DC サーバがどのキャッシュを保持しているかについて，ソフトウェアにより管理する必要がある．

実装では，Linux のページキャッシュの管理単位と同じく，DC サーバ上のキャッシュの管理単位を 4KB とした．これは ext3 ファイルシステムのブロックサイズに等しい．この場合，キャッシュは i-node とファイルオフセット値で管理が可能である．Linux2.6.15 の実装ではファイルオフセット値は 4KB ごとに 1 つの値が取られる．

DC サーバにキャッシュされているデータの管理には，ファイルオフセット値をノードとする木構造を利用する．本論文における実装では，Linux のページキャッシュ管理でも利用されている radix tree を用いた．

さて，この管理領域 (radix tree) の配置場所について以下の２通りが考えられ，それぞれに特有の問題がある．

（1） 管理領域を DC サーバ上に確保する

read/write の際に，キャッシュが存在するか DC クライアントから DC サーバに問い合わせる必要がある．あるいは，DC サーバにキャッシュが存在するものとして投機的にキャッシュの取得を試みることになる．しかし，DC サーバにキャッシュが存在しなかった場合，ネットワークパケットの往復時間により大幅に性能が低下するという問題がある．

（2） 管理領域を DC クライアント上に確保する

(1) の欠点は無くなるが，ローカルディスクキャッシュ領域やカーネルメモリ空間が管理領域により圧迫されるという問題がある．

DC クライアントが 32bit マシンの場合，現在の実装では DC サーバのキャッシュ1GB を管理するのにおよそ 8MB のメモリが必要である．本論文における実装では，(2) の DC クライアントに管理領域を保持する方式としている．なお，DC サーバへの接続時に DC サーバの管理に必要なだけのメモリ空間を予約する実装としている．

3.8 キャッシュアルゴリズム

本提案手法のキャッシュアルゴリズムを実装する際は，少なくとも以下の 3 点を考慮する必要がある．

（1） どのデータを DC サーバにキャッシュするか．

（2） DC サーバにデータがある時，DC サーバから読み込むか，ローカルディスクから読み込むか．

（3） キャッシュ領域が不足したとき，どのキャッシュデータを破棄するか．

(1)(2) に関しては，3.4 で述べた帯域の制限により，本手

法がシーケンシャルアクセス時に動作すると性能低下を引き起こすという問題を考慮する必要がある．

まず (1) については，ランダムアクセスされるファイルを優先的にキャッシュする必要がある．不必要に DC サーバへキャッシュを行うと，ディスクからの読み込みと DC サーバへの送信の 2 倍の I/O が発生し，システム性能が低下する．また同様に，(2) に関してもシーケンシャルリード時にはローカルディスクから読み込み，ランダムリード時には DC サーバから読み込む必要がある．通常のキャッシュシステムではキャッシュから読み込む方が高速であるため，(2) は本手法特有の考慮事項であるといえる．例えば OS のディスクキャッシュ機構では，ディスクから読むよりもローカル物理メモリから読み込んだ方が高速である．そのため，OS の場合はシーケンシャルリードされるファイルデータもローカル物理メモリキャッシュすることで性能向上を図れる．

本論文における実装では (1)(2) を解決するために，Linux の先読み機構と協調する動作とした．Linux はシーケンシャルアクセスと判定した場合，アプリケーションが要求したファイルオフセットよりも先まで読み込み命令を発行する．そこで，本論文における実装では，以下に示す戦略をとった．なお，n は DC サーバよりもローカルディスクから読んだ方が高速であると期待できるブロック数であり，DC 動作閾値と呼ぶ．DC 動作閾値はシステムに依存する変数であり，本提案手法の効果を最大化する n を利用する必要がある．

- n ブロック以上の先読み
  必ずディスクから読み込む．
- n ブロック未満の先読み or 単一ブロック読み込み
  - DC サーバにキャッシュされている場合
    DC サーバから読み込む．
  - DC サーバにキャッシュされていない場合
    ディスクから読み込み，Linux のディスクキャッシュにキャッシュし，DC サーバにもキャッシュする．
- 書き込み
  - DC サーバにキャッシュされている場合
    Linux のディスクキャッシュにキャッシュし，DC サーバのキャッシュを破棄する．
  - DC サーバにキャッシュされていない場合
    Linux のディスクキャッシュのみにキャッシュをする．

DC 動作閾値 n として，固定値を利用する方法，DC 動作閾値を動的に決定する方法，そして，ユーザがチューニングを行い決定する方法が考えられる．ユーザがチューニングを行わずとも最高の性能向上が得られるという目標のためには，DC 動作閾値を動的に決定するべきである．しかし，現在の実装では動的な DC 動作閾値の決定に対応していない．そこで，本論文における実験では DC 動作閾値として固定値 32 を用いた．これは，Linux のデフォルトでは先読みの最大値が 128KB であり，ブロックサイズ 4KB 換算で 32 ブロックが先読みの最大値だからである．すなわち，先読みの最大値がデフォルトの環境では，最大限に先読みが働いている場合，必ずローカルディスクから読み込むことになる．一方，書き込み時は DC サーバのキャッシュデータを破棄する実装となっている．

以上の実装から，DC サーバにデータがキャッシュされるのは，32 ブロック未満の読み込みが発生した時のみである．なお，ハードディスク上のデータ配置はファイルシステムに依存するという問題がある．そのため，ファイルオフセット値が連続していても，ランダムアクセスとなる可能性がある．この点を解決するためにはファイルシステムと協調した実装が必要であるが，本論文においてはファイルオフセット値のみでの実装としている．

(3) に関しては LRU を用いた．ここでの LRU とは，DC サーバから DC クライアントへのキャッシュ転送回数に基づく LRU である．つまり，最近 DC サーバから DC クライアントへ転送されていないキャッシュから順に破棄される．

3.9 プロトコル

本論文における実装では，データの正当性確保のため DC サーバと DC クライアント間のコネクションに TCP を利用した．UDP，その他のプロトコルを利用した場合の性能比較等は今後の課題とする．

DC サーバと DC クライアント間の通信は，キャッシュデータの送受信と，DC クライアントから DC サーバへのキャッシュ破棄指示である．キャッシュの破棄指示は，write された場合に発行される．

4. 評　　価

4.1 環　　境

本論文においては 1 台の DC サーバと 1 台の DC クライアントに限定して実験を行った．環境は，業務レベルの環境A (表 2) と個人レベルの環境B (表 3) との 2 種類である．以下のいずれの実験も「手法無効」の場合は公式の Linux カーネル 2.6.15 上で実験し，「手法有効」の場合は修正した Linux カーネル 2.6.15 上で実験した．

4.2 ディスクアクセスに対する評価

本節ではシーケンシャルとランダムなディスクアクセスに対する評価を行う．

4.2.1 シーケンシャルリード

実験環境 B におけるシーケンシャルリードの実験結果を表 4 と図 2 に示す．図 2 は本手法による性能向上比を示したものである．

この実験では，実験環境Bにおいて 64MB から 2GB までのファイル全領域に対し 2 回続けてシーケンシャルリードを行い，1 回目と 2 回目の実行時間を比較した．1 回目でキャッシュされ，2 回目でキャッシュヒットすれば 2 回目の性能は 1 回目と比較して向上する．2 回目でも性能向上がない場合は，キャッシュが溢れていることを示す．

本論文では，シーケンシャルリード時は動作しない実装としているため，提案手法による性能向上は得られない．すなわち，表 4 の 2 回目における性能向上はローカルディスクキャッシュによるものである．1 回目と 2 回目の結果を比較すると，提案

表 2　実験環境 A (業務レベル)

	DC クライアント (Dell PowerEdge 2850)	DC サーバ (Dell Precision 490)
CPU	Intel 64bit Xeon 3.2GHz (HT 有効) × 2	Intel Xeon 5110 × 2
Memory	DDR2 400 8GB (PAE36 使用)	DDR2 533 16GB (DC サーバ用に 12GB)
Disk	Ultra SCSI 320 15000rpm RAID5 (PERC 4e/Di Standard FW 521S DRAM:256MB)	SAS 15000rpm NoRAID
FS	ext3	ext3
OS	Linux 2.6.15 (x86 smp 公式版 or 修正版) (Fedora Core 5)	Linux 2.6.9-5.EL (x86_64 smp) (Red Hat Enterprise Linux WS Release 4)
Link	1000Base-T	1000Base-T
HUB	Dell PowerConnect 2716	

表 3　実験環境 B (個人レベル)

	DC クライアント	DC サーバ
CPU	AMD Geode NX 1500	Intel Pentium4 2.8CGHz
Memory	DDR333 512MB	DDR400 2GB (DC サーバ用に 1.2GB)
Disk	SATA 7200rpm NoRAID	SATA 7200rpm NoRAID
FS	ext3	ext3
OS	Linux 2.6.15 (x86 公式版 or 修正版) (Fedora Core 5)	Linux 2.6.15 (x86 smp) (Fedora Core 5)
Link	1000Base-T	1000Base-T
Hub	BUFFALO LSW-GT-5NS	

表 4　シーケンシャルリード 提案手法の効果 (実験環境 B)

アクセス容量	所要時間 [sec]			
	提案手法無効		提案手法有効	
	1 回目	2 回目	1 回目	2 回目
64MB	1.41	0.24	1.41	0.25
128MB	2.72	0.61	2.75	0.73
256MB	5.38	2.03	5.78	5.33
512MB	10.55	10.53	10.60	10.57
1GB	21.31	21.37	21.30	21.31
2GB	42.60	42.69	42.61	42.71

手法が無効の場合は 512MB，有効の場合は 256MB でローカルディスクキャッシュが溢れていることが分かる．提案手法が有効の場合に，無効の場合よりも少ない容量でローカルディスクキャッシュの溢れが起こったのは，3.7 で述べたとおり，DC サーバの管理に必要なメモリをローカルに確保するため，ローカルディスクキャッシュ領域が圧迫されたためである．

4.2.2　ランダムリード

ランダムリードの実験結果を表 5 と表 6 に示す．図 3 は，本手法による実験環境Bにおける性能向上比を示したものである．

この実験では，実験環境A・Bにおいて表 5 と表 6 の「アクセス容量」の大きさに等しいファイルに対し 2 回続けてランダムリードを行い，1 回目と 2 回目の実行時間を比較した．2 回目ではキャッシュヒットすれば性能が向上する．各回の実験は，4KB の倍数でファイル位置を選択し，選択した位置から 4KB の読み込みを行う．これを繰り返し，読み込んだ合計量が「アクセス容量」に等しくなるまで続ける．1 回目と 2 回目では同じ乱数列を用いており，アクセスパターンは等しい．また，乱数を用いたため，ある領域が 2 回以上読み込まれる可能性もある．なお，4KB は ext3 ファイルシステムのブロックサイズと等しい．

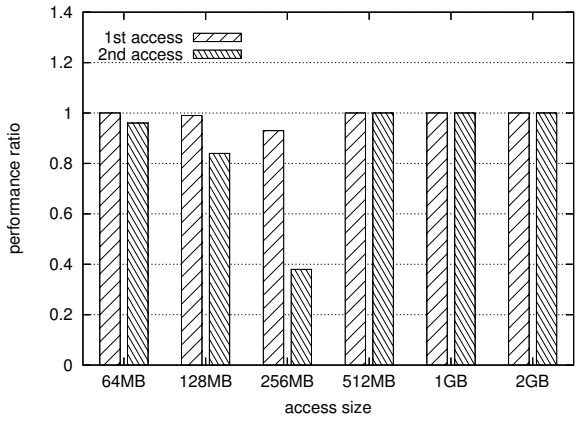

図 2　本手法によるシーケンシャルリード性能向上比 (実験環境 B)
(手法無効時を 1.00 とした)

表 6 のとおり，ランダムアクセス時は本手法により性能が向上する．アクセス量 1GB の 2 回目を手法無効時と有効時で比較すると，本手法により 10.3 倍の性能向上が得られていることが分かる．手法無効時のアクセス量 512MB 以上で 2 回目の性能が急激に低下したのは，ローカルディスクキャッシュが溢れたためである．ローカルディスクキャッシュが溢れるアクセス量 512MB 以上の場合は，乱数の重複のため，提案手法を用いると 1 回目でも高速化されている．なお，提案手法有効時にアクセス量 2GB で急激に性能が低下しているのは，DC サーバのキャッシュ(1.2GB) が溢れたためである．

4.3　データベースに対する評価

本論文ではデータベースに対してもベンチマークを行った．

表 5　ランダムリード 提案手法の効果 (実験環境 A)

アクセス容量	所用時間 [sec]			
	提案手法無効		提案手法有効	
	1 回目	2 回目	1 回目	2 回目
4GB	1630.65	2.87	1543.30	4.38
8GB	5898.31	5.60	3626.48	131.49
16GB	12255.99	6280.92	9500.07	946.26

表 6　ランダムリード 提案手法の効果 (実験環境 B)

アクセス容量	所用時間 [sec]			
	提案手法無効		提案手法有効	
	1 回目	2 回目	1 回目	2 回目
64MB	40.01	0.30	43.01	0.29
128MB	92.99	0.58	99.76	0.59
256MB	197.05	3.23	209.62	1.78
512MB	538.95	454.27	441.54	46.92
1GB	1447.30	1413.55	945.30	136.76
2GB	3451.03	3430.72	2390.32	1999.29

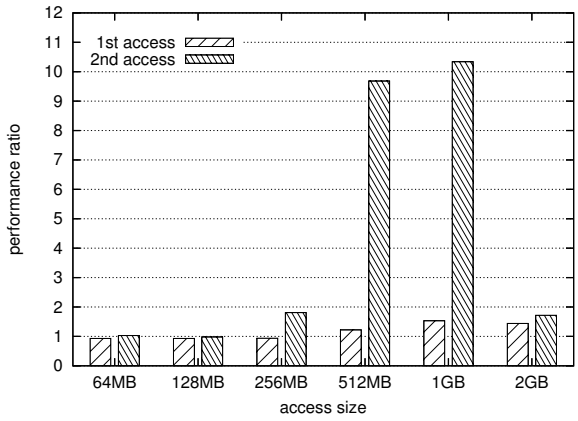

図 3　本手法によるランダムリード性能向上比 (実験環境 B)
(手法無効時を 1.00 とした)

ベンチマークでは，TPC-H [10] を参考に作成され，フリーで利用が可能な DBT-3 [11] を用いた．測定対象のデータベースは PostgreSQL [12] とした．組み合わせは，いずれも執筆時点で最新の DBT-3 1.9 と PostgreSQL 8.2.0 とした (表 7)．ベンチマークにあたっては OSS iPedia [13] で配布されているパッチ (dbt3-1.9-20060329.patch.gz) を適用した．

DBT-3 はロードテスト，パワーテスト，スループットテストの 3 種類のベンチマークを行う．ロードテストでは，データをデータベースに投入し，インデックスの作成を行う．ロードテストの結果は所要時間で示され，所要時間が短いほど高性能といえる．パワーテストでは，接続数 1 でデータベースへの問い合わせとデータの追加・削除を行う．スループットテストでは，複数接続でパワーテストと同様のテストが行われる．パワーテストとスループットテストの結果は 1 時間毎のトランザクション数×スケールファクタで示され，値が大きいほど高性能といえる．

DBT-3 が行うベンチマークの負荷レベルはスケールファクタで表される．スケールファクタはデータベースへ投入されるデータソースファイルの容量 (GB) である．本実験では，実験環境 A においてスケールファクタ 1 から 8，実験環境 B ではスケールファクタ 1 で測定を行った．また，手法有効時と無効時で公平な状態になるよう DBT-3 の乱数シードは同じ値を使用している．

表 7　データベース測定内容

データベースソフト	PostgreSQL
バージョン	8.2.0
ベンチマークソフト	DBT-3 1.9
スケールファクタ	1〜8
ストリーム数	2

表 8　DBT-3 による PostgreSQL 測定結果 (実験環境 A)

スケールファクタ	ロードテスト [sec]		パワーテスト		スループットテスト	
	提案手法の有効／無効					
	無効	有効	無効	有効	無効	有効
1	273	283	713.89	666.2	556.22	553.45
2	606	543	611.37	552.02	523.17	525.34
4	1169	1299	279.39	257.75	224.06	261.22
6	2231	2390	153.2	182.94	53.96	127.21
8	2590	3221	108.97	101.47	34.15	90.91

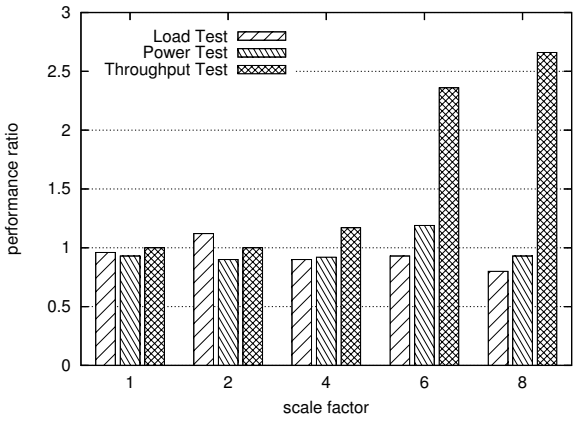

図 4　本手法による DBT-3 性能向上比 (実験環境 A)
(手法無効時を 1.00 とした)

表 9　表 8 の実験におけるキャッシュ転送量 (実験環境 A)

スケールファクタ	キャッシュ転送量 [GB]		
	To DC サーバ	From DC サーバ	破棄キャッシュ容量
1	0.01	0.00	0.00
2	0.29	0.06	0.04
4	8.36	0.40	1.67
6	17.22	93.02	6.43
8	22.25	193.82	10.25

4.3.1 考　　察

環境 A の実験結果を表 8 と図 4 に示す．キャッシュの転送量を表 9 に示す．図 4 は，本手法による性能向上比を示したものである．データベースでは書き込み処理や，シーケンシャルアクセスとランダムアクセスが複合的に発生するため，4.2 のよ

表 10　DBT-3 による PostgreSQL 測定結果 (実験環境 B)
(スケールファクタ：1)

	ロードテスト [sec]	パワーテスト	スループットテスト
手法無効	1114	52.71	7.98
手法有効	1147	59.72	24.54
性能向上	0.97	1.13	3.08

うな単純なベンチマークよりも性能向上が難しくなる．

提案手法を有効にした場合，ロードテストでは性能低下が発生した．ロードテストではテキストファイルからデータを読み込み，データベースへ投入する．テキストファイルからの読み込みはシーケンシャルリードであり，本手法の効果が得られない．また，インデックス作成における書き込み処理は，読み込み時のみキャッシュを行う本手法の実装により高速化がされない．パワーテストでは提案手法を有効にすると性能が最高 1.19 倍になった一方で，最低の場合では 0.90 倍になった．スループットテストでは，スケールファクタ 1 の時に 0.5%低下したが，スケールファクタ 8 の時には 2.66 倍の性能向上を得られた．スループットテストにおいてはデータベースへの接続数が増えるため，ランダムアクセスが増加し，本手法の効果が現れたからと考えられる．

環境 B の結果 (表 10) では，スループットテストにおいて，3.08 倍の性能向上が得られた．これは，本論文におけるデータベースに対するベンチマークでの最高性能向上率であった．

5. おわりに

5.1 まとめ

本論文では，ネットワークに接続されたリモートマシンのメモリを利用してディスクキャッシュの領域を拡張し，その性能を評価した．PostgreSQL に対する DBT-3 によるベンチマークでは，環境 B におけるスループットテストで最大 3.08 倍の性能向上を達成した一方で，環境 A におけるロードテストでは最大 24%の性能低下があった．

5.2 本手法の想定される利用法

本論文の実験では，1 台の DC サーバと 1 台の DC クライアントを接続している．また，DC サーバのメモリ量は DC クライアントよりも多くなっている．このような環境を用意できるなら，DC サーバを運用に回した方が良い．本手法が想定するのは，既存のシステムに余剰資産のマシンを DC サーバとして接続することでキャッシュ容量を確保するという利用法である．最大容量のメモリを搭載している場合や，システム構成を変更するのが困難な場合でも，性能向上の最終手段としての利用が可能であると考える．

5.3 課　題

本論文における実装では信頼性を確保するため TCP を利用したが，UDP などの異なるプロトコルの利用も検討が必要である．また，本手法はネットワーク通信を大量に行うため CPU 負荷が増加する．DBT-3 によるベンチマークでは Disk I/O の時間が多く，CPU に余裕があるため問題とはならなかったが，CPU 負荷が高い環境では問題となる可能性がある．

DBT-3 によるベンチマークでは，ロードテストを始めとして，一部のベンチマークでは性能が低下している．今後，実装の検討を行い，さらなる高速化を目指す．また，DC 動作閾値を動的に決定することで，より性能を向上させることも目標とする．

現在，実験データの量が不充分であり，提案手法の有効性の評価が充分にできていない．同じ理由で，実験結果に対する検討も困難な点がある．また，複数台の DC サーバに複数台の DC クライアントを接続するといった形態での実験も必要である．実験データを充実させ，今後の会議において報告する予定である．そして，最終的には本提案手法を利用するためのカーネルパッチの配布も考えている．

謝辞　査読者の方からは貴重なご意見を頂きました．ここに感謝の意を表します．

文　献

[1] D. Comer and J. Griffioen,“ A New Design for Distributed Systems: The Remote Memory Model, ”In Proceedings of the USENIX Summer 1990 Technical Conference, pp. 127-136, June, 1990.
[2] Liviu Iftode, Kai Li, Karin Petersen,“ Memory Servers for Multicomputers, ”In Proceedings of the IEEE Spring COMPCON ’93, pp.538-547, February 1993.
[3] Tia Newhall, Sean Finney, Kuzman Ganchev, Michael Spiegel,“ Nswap: A Network Swapping Module for Linux Clusters, ”In Proceedings of Euro-Par International Conference on Parallel and Distributed Computing, vol.2790, August 2003.
[4] Kai Li, Karin Petersen,“ Evaluation of Memory System Extensions. ”In Proceedings of the 18th annual International Symposium on Computer Architecture, pp. 84-93, 1991.
[5] C. Amza, A.L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel,“ TreadMarks: Shared Memory Computing on Networks of Workstations, ”In IEEE Computer, vol.29, no.2, pp.18-28, February 1996.
[6] Microsoft, http://www.microsoft.com/.
[7] Windows PC Accelerators, http://www.microsoft.com/whdc/system/sysperf/perfaccel.mspx.
[8] Intel, http://www.intel.com/.
[9] Michael Trainor,“ Overcoming Disk Drive AccessBottlenecks with Intel Robson Technology, ”Technology@Intel Magazine, vol.4, no.9, December 2006.
[10] Transaction Processing Performance Council, http://www.tpc.org/.
[11] OSDL Database Test 3, http://www.osdl.org/lab_activities/kernel_testing/osdl_database_test_suite/osdl_dbt-3/.
[12] PostgreSQL: The world’s most advanced open source database, http://www.postgresql.org/.
[13] OSS iPedia, http://ossipedia.ipa.go.jp/.