2007－ARC－174 (26)
2007／8／2

リモートメモリを用いたランダムディスクアクセス高速化手法

上田 高徳[†] 平手 勇宇[†] 山名 早人[††,‡]

† 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

コンピュータシステムの性能向上のためには，ハードディスクアクセスの高速化が欠かせない．しかし，ハードウェアやアプリケーションの修正が必要となる高速化手法は導入コストが高い．そこで本論文では，ハードウェアとアプリケーションの修正が不要な，OS レベルで実現できる高速化手法を提案する．具体的には，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ディスクキャッシュに用いることでディスクアクセスの高速化を図る．本論文では提案手法を Linux Kernel 2.6 に実装し，DBT-3 による PostgreSQL に対するベンチマークを行ったところ，物理メモリを提供するマシンの台数が 2 台の時に 1.52 倍，4 台の時に 3.10 倍，8 台の時に 6.68 倍の高速化効果が得られた．

Exploiting Remote Memory to Speed-up Random Disk Access

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

† 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

As hard disks tend to be bottleneck devices in the current computer architecture, enhancing hard disk access speed is one of the most efficient factors to improve the total performance of computers. However, hardware modification with or without software modification is costly. Accordingly, we propose a new acceleration method implemented at the OS kernel level, which has no requirement for modification of hardware or existing applications. Specifically, our method exploits remote memory to extend local disk cache. We have implemented our method on Linux Kernel 2.6. The PostgreSQL benchmark results show that our method increases computer speed by 1.52 times when using two remote machines, 3.10 times when using four remote machines, and 6.68 times when using eight remote machines.

1. はじめに

ハードディスクは大容量のデータを扱うために欠かせないが，そのアクセス速度の低速さが原因でシステム性能を低下させることが多い．特に，ファイルへのランダムアクセス時は，ハードディスクのアクセスヘッドを物理的に動かす必要があるため，アクセス速度が著しく低速となり，システムの性能が大きく低下する．また，マルチ CPU 環境では，並列動作するアプリケーションによりランダムアクセスが多発し，ますますハードディスクがシステム性能のボトルネックになると考えられる．今やディスクアクセスの高速化はコンピュータシステムの高速化のために欠かせない．

ハードディスクの高速化は，アーキテクチャのどのレベルで実現する手法かによって，以下の３通りに分類できる．

- ハードウェアレベルの高速化手法[6][9][11][15]
- アプリケーションレベルの高速化手法[17]
- OS レベルの高速化手法[2][3][4][5][7][8][15]

ハードウェアレベルの高速化手法とは，ハードウェア上の拡張を行うことで高速化を図るものである．ハードでしか知り得ない情報を用いて高速化が可能である．例えば，アクセスヘッドの現在位置をコントロールすることで高速化する研究[6][9]などがある．しかし，ハードウェアの修正が必要な手法はコストが高い．何よりも，ハードウェアの置き換えを伴う場合，既存システムへの導入が困難という問題がある．

アプリケーションレベルの高速化手法とは，アプリケーションを修正してソフトウェア的に高速化手法を実装するものである．アプリケーションのアクセス特性に適した高速化を行うことができる[17]．しかし，個々のアプリケーションを修正する必要がある上に，高速化の効果を得られるのは修正を行ったアプリケーションのみとなる．

そこで，OS レベルにディスクアクセス高速化手法を実装することが考えられる[2][3][4][5][7][8][15]．OS レベルの修正であれば，既存ハードウェア及び既存アプリケーションの変更をしなくとも，システムの高速化が可能となる．

我々は，OS レベルの高速化手法として，ネットワーク上のリモートマシンの物理メモリ，いわゆるリモートメモリをローカルディスクキャッシュとして利用することを提案している[16]．本システムは，ランダムアクセスされるファイルをリモートメモリにキャッシュすることで高速化を行う．本システムは Linux 2.6 カーネルを修正することで実装されている．OS レベルでの実装のため，ハードウェアや既存アプリケーションソフトの修正・再コンパイルは不要である．本論文では，[16]において未評価であった，メモリ提供側

のマシンが複数台の場合における実験結果を述べる．また，[16]における実装を改良して再実験した結果についても述べる．

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述べる．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

本節では，リモートメモリをディスクアクセスの高速化に利用する研究[3][4][5][7][8]と，ディスクキャッシュの拡張のためにローカルデバイスを用いる製品[11][15]について述べる．

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リソースとして有効利用することは，[2]において“Remote Memory Model”と名付けられている．これに倣い，以下リモートマシンの物理メモリのことをリモートメモリと呼ぶ．“Remote Memory Model”の考え方のひとつは，ローカルディスクに対するランダムアクセスよりも，リモートメモリに対するランダムアクセスの方が高速なことをシステムの性能向上に利用することである．

例えば，Memory Servers[4]やAnemone[5]は，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を行っている．また，Nswap[7]は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ることで，スワップ発生時のクラスタシステム高速化を行っている．SDNMS[3]は，リモートメモリをスワップもしくはストレージとして利用することで性能向上を達成している．RiCache[8]はiSCSIディスクのデータをInfiniBandで接続されたリモートメモリにキャッシュすることで性能向上を図っている．

これらの研究は，リモートメモリをハード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．しかし，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，スワップ動作の高速化が直ちにシステムの性能向上には繋がる訳ではない．そして，リモートメモリを単にストレージとして利用した場合は，ストレージ容量がリモートメモリの総容量に制限されることになる．また，RiCache[8]はiSCSI環境を前提にしており，ローカルディスクアクセスの高速化を行っていない．

2007年に入り，ローカルに接続されたデバイスを用いてディスクキャッシュ領域を拡張し，ディスクアクセスを高速化する製品が普及し始めている．Microsoft ReadyBoost[15]はWindows Vistaに搭載された新機能であり，USBメモリをディスクキャッシュ領域の拡張に利用する．Microsoft ReadyDrive[15]はディスクドライブ本体に大容量の不揮発性メモリを搭載し，ディスクと不揮発性メモリをハイブリッドに使用する．また，Intel Turbo Memory [11]は，PCI Expressのハードディスク専用キャッシュカードを装着する方式である．ReadyBoostとReadyDriveはOSと連携して動作するため，Windows Vistaにおける利用が前提となっている．Intel Turbo Memoryは2007年に発売されたIntelの新チップセットを搭載したマザーボードでなければ利用できない．

本論文の手法は，リモートメモリをローカルディスクキャッシュの拡張領域として利用する．ディスクキャッシュ領域の拡張に利用すれば，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．また，キャッシュとしてリモートメモリを利用するため，リモートメモリの容量にストレージ容量が制限されることはない．さらに，ハードディスクが苦手なランダムアクセスされるデータをリモートメモリにキャッシュするため，全てのデータをキャッシュするよりも効率が良い．環境の変更を余儀なくされる製品[11][15]に対し，本論文の実装では，既存環境がLinuxであれば，Linuxカーネルを入れ替えることで導入が可能である．

3. 提案手法

本節では，提案手法の詳細について述べる．

3.1. 提案手法の概要

提案手法では，OSが標準で持つローカルメモリ上のディスクキャッシュに加え，リモートメモリもディスクキャッシュとして利用することで高速化を図る．また，データの損失を防ぐため，リードデータのみをリモートメモリにキャッシュする．図1に提案手法の概要図を示す．以下，ディスクキャッシュ用に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）サーバと呼ぶ．また，DCサーバ上のメモリをディスクキャッシュとして利用するマシンをDCクライアントと呼ぶ．図1のとおり，複数台のリモートマシンをDCサーバとして利用可能である．

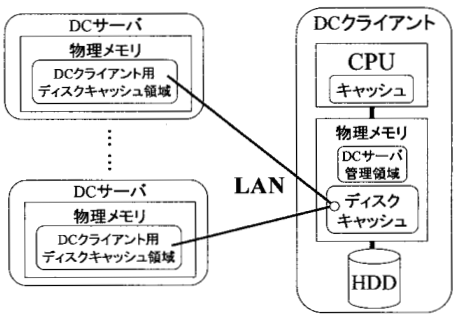

図 1：提案手法の概要図

3.2. 高速化が可能な理由

本論文の実験では，DCサーバとDCクライアント間のネットワークとして1000Base-Tを利用し，プロトコルとしてTCPを用いた．1000Base-Tの帯域は，ハードディスクのインターフェース（Serial ATA，SCSI）よりも低速であるが，ハードディスクはシークタイムによるオーバーヘッドがあるため，リモートメモリに

対するランダムアクセスはローカルディスクに対するランダムアクセスよりも高速である．ハイエンドのハードディスクであっても平均シークタイムが 3ms[10]であるのに対し，LAN 内通信のレイテンシは，数百マイクロ秒である．

そこで本システムは，ランダムアクセス時にキャッシュ動作を行うことで，性能の向上を達成する．

3.3. 現在の実装

提案手法ではディスクキャッシュ機構を修正するため，OS レベルの修正が必要である．本論文では実装と評価の対象として Linux 2.6 カーネルを選択した．OS レベルの実装のため，DC クライアントへの導入は Linux カーネルの入れ替えで行え，既存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．

メモリ提供側である DC サーバのプログラムはユーザプログラムとして実装を行える．従って，DC サーバの OS には多くの選択肢がある．本論文の実験では Linux を用いている．

3.3.1. 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．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 クライアント間の接続に TCP を利用した．また，完全に接続が失われた際のデータ損失を防ぐため，本論文における実装では write キャッシュは行わない．

セキュリティ対策のためには通信の暗号化を行う必要がある．しかし，現在のところ DC サーバ用に専用のネットワークを構築して運用することを前提としているため，暗号化は行っていない．

3.3.2. DC サーバが複数台の場合

本手法では，複数台のリモートマシンを DC サーバとして利用可能である．複数台の DC サーバを利用する場合，現在のシステムではリモートメモリを１つのメモリ空間とみなす．言い換えれば，1GB の DC サーバが 8 台の場合と，8GB の DC サーバが 1 台の場合で，キャッシュアルゴリズム上に違いは発生しない．

3.3.3. キャッシュ管理

提案手法では，どの DC サーバがどのデータをキャッシュしているか管理する必要がある．現在の実装では，キャッシュの単位を 4KB とし，ファイルの i-node とファイルオフセットでキャッシュ管理を行う．キャッシュされているか否かを高速に判定するため，本論文における実装では Linux のディスクキャッシュ管理でも利用されている Radix Tree を用いた．

管理領域（Radix Tree）の配置場所については，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：
DC クライアントのメモリを節約できる．しかし，read / write の際に，「キャッシュが存在するか」について 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．言い換えれば，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．

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．パケットの往復時間を節約するため，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．また，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．

3.3.4. キャッシュアルゴリズム

本手法のキャッシュアルゴリズムを実装するにあたり，以下の３つの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のか．

(2) DC サーバにデータがキャッシュされているとき，DC サーバとローカルディスクのどちらから読み込むか．

(3) DC サーバのキャッシュ領域が溢れたとき，どのキャッシュデータをリプレースするか．

3.2で述べたように，ハードディスクはシーケンシャルアクセスが高速であることを考慮すると，(1)に関しては，ランダムアクセスされるデータのみを DC サーバにキャッシュするべきである．ハードディスクはシーケンシャルアクセスが高速であるから，ネットワークの帯域を考えると，シーケンシャルアクセスされるデータをキャッシュする利点が無い．(2)も同様に，DC サーバにデータがキャッシュされていたとしても，シーケンシャルアクセス時はローカルディスクから読むべきである．さらに(3)に関しては，最近ランダムアクセスされていないデータから破棄するべきである．

現在の本システムは Linux の先読み機構と協調して動作する．Linux はシーケンシャルアクセスが発生している限り先読みを行い，先読みサイズを指定された最大先読みサイズまで増加させる[1]．デフォルトの設定では，128KB までの先読みを行う．そこで本システムは，デフォルトの最大先読みサイズである 128KB を閾値として，ランダムアクセスとシーケンシャルアクセスを識別する．

一度ランダムアクセスされたデータは，再びランダムアクセスされる可能性が高い．従って，本システムはランダムリード時，すなわちディスクに対する読み込みリクエストが 128KB 未満の時に DC サーバにキャッシュを行う．また，DC サーバから読み込むのは，ランダムリード時，すなわち読み込みリクエストが 128KB 未満の時に限り，シーケンシャルリードが続いている際はローカルディスクから読み込む．

キャッシュリプレースアルゴリズムは，DC サーバから DC クライアントへの転送回数に基づく LRU とし

た．DC サーバが複数台の場合でも，全ての DC サーバの中から，最後に転送されてから最も時間が経過したキャッシュデータ（4kB 単位）を選択しリプレースする．アルゴリズムは単なる LRU であるが，DC サーバのキャッシュを読み込むのはランダムアクセス時のみであるので，結果として，前述した「最近ランダムアクセスされてないデータから破棄するべき」という条件を満たせることになる．

なお，Linux はローカルの空きメモリをディスクキャッシュとして最大限利用する[1]．本手法でも DC サーバにキャッシュすると同時に，Linux が持つローカルメモリ内のディスクキャッシュにもキャッシュする．

以上のキャッシュアルゴリズムを図 2に示す．図 2 での LRU リストとは，DC サーバにキャッシュされたデータに対する LRU リストである．なお，DC サーバにキャッシュ済みのデータに書き込みが発生した場合は，DC サーバ上のキャッシュを破棄する．

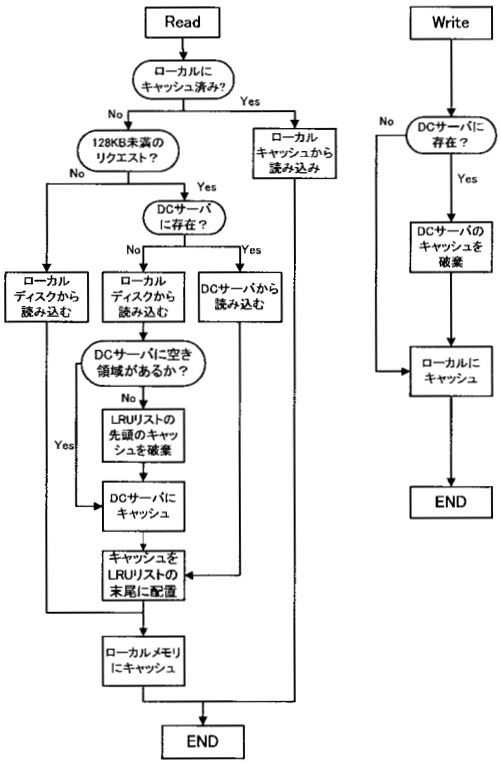

図 2：キャッシュアルゴリズム

4. 評価

本節では，提案手法の効果を確認するため，シーケンシャルリードとランダムリードによるベンチマーク，及びデータベースに対するベンチマークの結果について述べる．

4.1. 評価環境

本論文では，DC サーバと DC クライアントが一対一である環境A（表 1）と，DC クライアント 1 台に対し DC サーバが複数台である環境B（表 2）を用いた．環境Aでは，DC サーバは 12GB のメモリを DC クライアントに提供する．環境Bでは，DC サーバ 1 台あたり，512MB のメモリを DC クライアントに提供する．環境Aは[16]での実装を改良しての再実験となる．

表 1：評価環境 A

	DC Client (Dell PowerEdge 2850)	DC Server (Dell Precision 490)
CPU	Intel 64bit Xeon 3.2GHz (HT enabled) ×2	Intel Xeon 5110 ×2
Memory	DDR2 400 8GB (PAE36 有効)	DDR2 533 16GB (うち DC クライアントに 12GB 提供)
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) (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 2：評価環境 B

	DC Client・DC Server 共通
CPU	Intel Pentium 4 2.4GHz (HT 非対応)
Memory	SDRAM 1GB (うち DC クライアントに 512MB 提供)
Disk	Seagate Barracuda 7200.8　400GB
FS	ext3
OS	Linux 2.6.15 (Fedora Core 5)
Link	1000Base-T
HUB	NETGEAR GS524T

4.2. ファイルベンチマーク

本システムの基本性能を計測するため，本論文では，以下の 2 つの単純なベンチマークを，公式カーネルと提案手法を実装した修正カーネルのそれぞれにおいて行い，実行時間を比較した．

- シーケンシャルリードベンチマーク
  ファイルの全領域に対するシーケンシャルリードを 2 回続けて行い，1 回目と 2 回目の実行時間を比較する．本論文では，シーケンシャルリード時は提案手法が動作しない実装としているため，2 回目の実行時間短縮はローカルメモリ上のディスクキャッシュによる．ただし，修正カーネルでは，3.3.3 で述べた DC サーバ管理領域により，ローカルメモリ上のディスクキャッシュ容量が少なくなる．従って，本ベンチマークにより，DC サーバ管理領域のオーバーヘッドを計測することができる．
- ランダムリードベンチマーク
  ファイルに対するランダムリードを 2 回続けて行い，1 回目と 2 回目の実行時間を比較する．1 回目，2 回目とも 4KB ブロックずつランダムに読み込み，読み込んだ合計量がファイルサイズに等しくなるまで続ける．ブロックの選択には 1 回目と 2 回目で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，乱数を用いてブロックを選択するため，あるブロックが 2 回以上読み込まれる可能性がある．提案手法はランダムリード時に動作するので，本ベンチマークで提案手法の効果を確認できる．

4.2.1. 一対一接続（環境A）の場合

環境Aにおけるシーケンシャルリードベンチマークの，修正カーネルと公式カーネルの性能比を図 3に示す．図 3において 0.4 とは，修正カーネルにおいては，公式カーネルに対して 60%の性能低下が発生したことを示す．8GB 付近での急激な性能低下は，DC サーバ管理領域により，ローカルメモリ上のディスクキャッシュの溢れが，公式カーネルよりも小さなファイルサイズで始まり，2 回目のアクセスでディスクから読み込む必要が生じたのが原因である．すなわち，図 3の性能低下が発生している部分の幅が，DC サーバ管理領域として確保されている容量である．この現象は，特定のファイル容量で発生する問題であるため，大きな問題にはならないと考えられる．

環境Aにおけるランダムリードベンチマークの結果を表 3に示す．アクセス量が 16GB の 2 回目に 6.64 倍の性能向上を達成していることが分かる．乱数の重複により，複数回読み込まれるブロックがあるため，ローカルメモリのディスクキャッシュが溢れる 8GB 以上の時には 1 回目のアクセスでも高速化される．

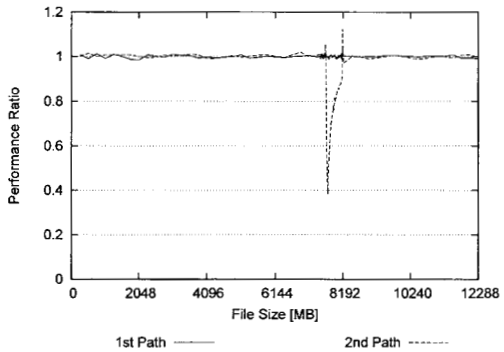

図 3：シーケンシャルリードベンチマークにおけるオーバーヘッド（環境 A）

表 3：ランダムリードベンチマーク結果（環境 A）

ファイルサイズ	実行時間[sec]			
	公式カーネル		修正カーネル	
	1 回目	2 回目	1 回目	2 回目
4GB	2075.95	3.23	2088.91	3.24
8GB	5898.31	5.60	5084.13	6.74
16GB	12255.99	6280.92	9500.07	946.26

4.2.2. DC サーバが複数台の場合（環境B）

DC サーバが複数台の場合のシーケンシャルリードベンチマークは，DC サーバの台数が増えると管理領域によるオーバーヘッドが大きくなる点を除けば環境 A と同様であるので，環境 B については，ランダムリードベンチマークの結果のみを示す．

環境 B では，ファイルサイズを 4GB に固定してランダムリードベンチマークを行った．結果を表 4に示す．2 回目のアクセスでは，DC サーバが 4 台の時に 1.49 倍，8 台の時に 29.08 倍の性能向上を確認できた．1 回目でも高速化されるのは，4.2.1と同様に，複数回読み込まれるブロックがあることによる．

表 4：ランダムリードベンチマーク結果（環境B）（ファイルサイズ 4GB）

DC サーバの台数	実行時間[sec]		性能向上比	
	1 回目	2 回目	1 回目	2 回目
0 (*)	6690.18	6422.79		
2	6424.04	5935.27	1.04	1.08
4	5381.87	4318.60	1.24	1.49
8	5394.65	220.86	1.24	29.08
16	5377.01	221.75	1.24	28.96

(*) 公式カーネル

4.3. データベースベンチマーク

本節では，フリーで利用が可能な DBT-3[12]を用い，PostgreSQL 8.2.0[13]に対するベンチマークを行った結果について述べる．DBT-3 は TPC-H[14]を参考に作成されており，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．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．

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 2 で測定を行った．

4.3.1. 一対一接続（環境A）の場合

環境 A での，修正カーネルによる DBT-3 性能向上比を図 4に示す．スケールファクタが 8 の時に，パワーテストで 1.57 倍，スループットテストで 2.36 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，本手法の効果が大きくなったからと考えられる．

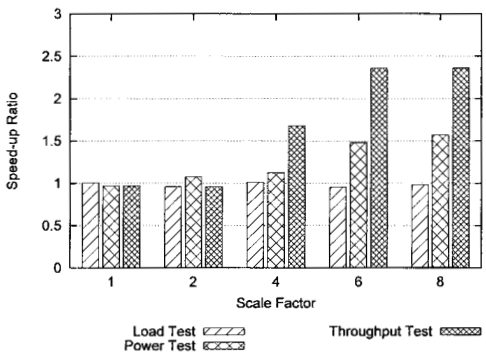

図 4：DBT-3 ベンチマーク性能向上比（環境 A）

ロードテストでは，最大4%の性能低下が発生した．ロードテストでのテキストファイルからの読み込みは1回のみの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．なお，ロードテストはデータベースの初期化に相当するベンチマークであり，この性能低下は実用上大きな問題にはならないと考えられる．

4.3.2. DC サーバが複数台の場合（環境B）

DC サーバが複数台の環境Bでは，スケールファクタを2としてDBT-3ベンチマークを行った．修正カーネルによる性能向上比を図 5に示す．パワーテストでの最大性能向上は，DC サーバの台数が 16 台の時の1.45倍であった．スループットテストにおいては，DC サーバの台数が2台で1.52倍，4台で3.10倍，8台で6.68倍，16台で6.81倍，そして32台で5.44倍の性能向上が得られた．DC サーバの台数が 32 台の時に 16 台の時よりも性能が低いのは，DC サーバの管理領域が原因と考えられる．なお，ロードテストでは，DC サーバが32台の時に9%低下したのが最悪のケースであった．

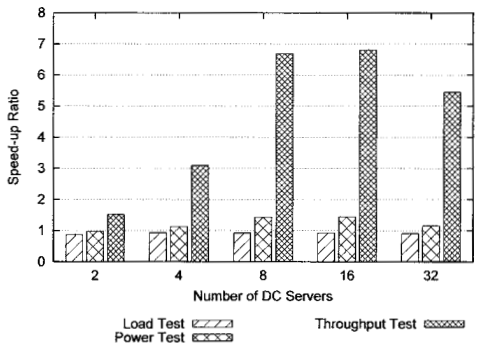

図 5：**DBT-3** ベンチマークにおける **DC** サーバの台数効果（環境B・スケールファクタ **2)**

5. おわりに

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．本手法を用いることで，既存のシステムに DC サーバを接続しキャッシュ容量を確保することが可能となる．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適用が可能である．

PostgreSQL に対する DBT-3 によるベンチマークでは，DC サーバ，すなわちメモリを提供するマシンが8台の時，スループットテストで6.68倍の性能向上を達成した．本手法の高速化対象ではないロードテストでは，DC サーバが32台の時に9%の性能低下があった．今後，実装の改良を行い，さらなる性能向上と，高機能化を目指す．

参 考 文 献

[1] D.P. Bovet, M. Cesati, *Understanding the Linux Kernel, Third Edition*, O'Reilly, Nov. 2005.

[2] D. Comer, and J. Griffioen, "A New Design for Distributed Systems: The Remote Memory Model", In *Proc. of the USENIX Summer 1990 Tech. Conf.* (Anaheim, CA), June 1990, pp.127-136.

[3] G. Sun, H. Tang, and M. Chen, "A scalable dynamic network memory service system", In *Proc. of the 8th Int'l Conf. on High-Performance Computing in Asia-Pacific Region* (HPCASIA'05, Beijing, China), Nov. 2005.

[4] L. Iftode, K. Li, and K. Petersen, "Memory Servers for Multicomputers", In *Proc. of the 38th IEEE Computer Society Int'l Conf.* (Compcon Spring'93, San Francisco, CA), Feb. 1993, pp.538-547.

[5] M.R. Hines, J. Wang, K. Gopalan, "Distributed Anemone: Transparent Low-Latency Access to Remote Memory", In *Proc. of Int'l Conf. on High-Performance Computing* (HiPC, Bangalore, India), Dec. 2006.

[6] R.M. English and A.A. Stepanov, "Loge: a self-organizing disk controller", In *Proc. of the USENIX Winter 1992 Tech. Conf.* (San Francisco, CA), Jan. 1992, pp.237-251.

[7] T. Newhall, S. Finney, K. Ganchev, and M. Spiegel, "Nswap: A Network Swapping Module for Linux Clusters", In *Proc. of the Euro-Par 2003 Int'l Conf. on Parallel and Distributed Computing* (Klagenfurt, Austria), Aug. 2003, pp.1160-1169.

[8] X. Liu, N. Wang, G. Sun, J. Han, L. Zhang, and C. Han, "Remote iSCSI Cache on InfiniBand: An Approach to Optimize iSCSI System", In *Proc. of the 2006 Int'l Conf. on Parallel Processing Workshops* (ICPPW'06, Columbus, OH), Aug. 2006.

[9] Y. Hu and Q. Yang, "DCD—disk caching disk: a new approach for boosting I/O performance", In *Proc. of the 23rd Ann. Int'l Symp. on Computer Architecture* (ISCA'96, Philadelphia, PA), May 1996, pp.169-178.

[10] Cheetah 15K.5 Data Sheet, http://www.seagate.com/docs/pdf/datasheet/disc/ds_cheetah_15k_5.pdf, Seagate Technology, May 2006.

[11] Intel Turbo Memory, http://www.intel.com/design/flash/nand/turbomemory/index.htm.

[12] OSDL Database Test 3, http://www.osdl.org/lab_activities/kernel_testing/osdl_database_test_suite/osdl_dbt-3/.

[13] PostgreSQL: The world's most advanced open source database, http://www.postgresql.org/.

[14] Transaction Processing Performance Council, http://www.tpc.org/.

[15] Windows PC Accelerators, http://www.microsoft.com/whdc/system/sysperf/perfaccel.mspx.

[16] 上田高徳，平手勇宇，山名早人，"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，" 電子情報通信学会 第18回データ工学ワークショップ（DEWS2007，広島），Mar. 2007.

[17] 深山辰徳，杉田秀，蛭田智則，山名早人，"先読みスレッドを用いたDiskアクセスの高速化，" 情報処理学会研究報告，2007-ARC-172，Vol.2007，No.17，pp.233-238.