情報処理学会論文誌　コンピューティングシステム　Vol. 4　No. 4　228–245 (Oct. 2011)

# 仮想マシンモニタによる透過的ネットワークブート方式

表　祐志[†1]　品川高廣[†1,∗1]　加藤和彦[†1]

近年，ネットワークブート方式を用いてストレージをサーバで集中管理するシステムが広く普及している．しかし，従来のネットワークブート方式では，サーバのストレージにアクセスするために OS に依存した方式が必要であったり，クライアント端末の機能が制限されたりするなど，OS やユーザから見て通常の PC と同様に見えるような透過的なシステムの実現は難しい．本論文では，仮想マシンモニタを用いた透過的なネットワークブート方式を提案する．OS が通常の PC と同様の方式でサーバのストレージにアクセスできるようにするために，仮想マシンモニタでストレージデバイスへのアクセスを捕捉し，ネットワークを介したサーバへのアクセスに変換する．また，クライアント端末の機能を通常の PC と同様に使えるようにするために，ストレージとネットワークデバイス以外のデバイスへのアクセスをパススルーとする．これにより，既存の OS をいっさい改変せずにネットワークブートできるほか，クライアント端末の内蔵機器や周辺機器を通常の PC と同様に活用できる．提案方式を仮想マシンモニタである BitVisor をベースに実装し，動作確認実験により既存の PC 向けの Linux ディストリビューション（Debian）や Windows（Vista，7）をいっさい改変せずにネットワークブートできることを確認した．また，性能評価の結果，提案方式での OS の起動時間は，サーバ OS のキャッシュが有効な場合は，ローカルディスクから起動した場合と比べて，14 秒程度短縮された．キャッシュの影響を除いても，24 秒程度の増加に抑えられた．また，アプリケーションベンチマークの結果，メモリ管理が大量に発生するユースケースを除き，OS をローカルディスクから直接起動した場合と同等の評価結果を得た．

# A VMM-based Transparent Network Boot System

Yushi Omote,[†1] Takahiro Shinagawa[†1,∗1]
and Kazuhiko Kato[†1]

Recently, network boot systems managing storage on servers are widely adopted because of its benefits of management cost reduction. However, existing network boot systems either require OS-specific technologies and configuration or limit the functionalities of devices on client computers, preventing the realization of transparent systems that just look like ordinary PCs. In this paper, we propose a VMM-based transparent network boot system. Our system allows the operating system to access storage on servers the same way with local disks by intercepting access to storage devices by the VMM and transforming them to access to storage on servers. Also, our system allows the operating system to utilize devices on client computers the same way with ordinary PCs by allowing pass-through access to the devices except for a storage device. We implemented our system based on a VMM called BitVisor, and our functional test confirmed that an Linux distribution for standard PC (Debian) and Windows (Vista, 7) can be network-booted without any modification. Our experimental results of measuring boot time showed that our system took approximately 14 seconds shorter than the local PC without network boot because of the cache effect on server OS.

## 1. はじめに

近年，ネットワークブート方式のシンクライアントが広く普及している．ネットワークブート方式とは，OS やアプリケーションなどソフトウェア資源をサーバに置き，クライアント端末からネットワーク経由で読み込んで起動する方式である．ネットワークブート方式の利点としては，ソフトウェア資源をサーバで集中管理することにより，管理コストを削減できる点があげられる[1)]．たとえば，OS やアプリケーションのインストールやバージョンアップ，セキュリティパッチの適用，バックアップ，障害復帰などのシステム管理作業をサーバで完結させることができるため，クライアント端末を個々に管理しなくてもよい．こうした利点が着目され，ネットワークブート方式のシンクライアントは多くの企業や教育機関などで導入されている．

本研究の目的は，ソフトウェア資源はサーバで集中管理しつつ，クライアント端末ではなるべく通常の PC と同等の環境を提供できるネットワークブート方式の実現である．たとえば，クライアント端末において通常の PC と同等の動作環境を実現することにより，OS やアプリケーションをいっさい修正することなくネットワークブートが可能になり，複雑なシステム管理作業を通常の PC の場合と同様のやり方で行うことができる．また，クライアン

†1 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Systems and Information Engineering, University of Tsukuba
∗1 現在，東京大学情報基盤センター情報メディア教育研究部門
Presently with Campuswide Computing Research Devision, Information Technology Center, The University of Tokyo



© 2011 Information Processing Society of Japan

ト端末のハードウェアが通常の PC と同等に機能することにより，ユーザや OS がハードウェアの機能をすべて利用できるほか，OS の資源管理や電源管理なども最も効率良く行えるため，クライアント端末のハードウェアを最大限に有効活用することができる．さらに，実行速度や応答性などの点でも，なるべく通常の PC と同等の性能を実現することが望ましい．すなわち，(1) 動作環境，(2) 機能，(3) 性能の 3 つの観点で，管理者やユーザ，OS やアプリケーションなどから透過的なシステムの実現を目指している．

従来の OS によるネットワークブート方式では，OS やアプリケーションがクライアント端末のハードウェア上で直接動作するため，通常の PC と同等の機能や性能を実現することは比較的容易である．しかし，Windows や Linux をはじめとするクライアント向け OS の多くはローカルディスクからのブートを前提とした構成となっているため，それらの OS でネットワークブートを実現するためには OS やディストリビューションに依存した複雑な設定を個々の管理者が個別に行う必要がある[2),3)]．また，バージョンアップや障害復帰などのシステム管理においてローカルディスクからブートした場合と異なる作業が発生する可能性があり，透過的な動作環境は必ずしも実現されていない．クライアント端末で仮想マシンモニタを動作させ，ゲスト OS を修正せずにネットワークブートできる方式も提案されているが[1),4)]，従来の汎用仮想マシンモニタでは計算機全体が仮想化されてしまうため，ユーザやゲスト OS がクライアント端末のハードウェアのすべての機能を利用することは難しい．また，仮想化によりゲスト OS がハードウェアの特性を活用した資源管理や OS の管理情報に基づく適切なタイミングでの電源管理などを効果的に行うことが難しくなるほか，様々なオーバヘッドが発生するなど，機能や性能の点で必ずしも透過的でない．グラフィックスなど一部のデバイスに対してゲスト OS が直接アクセスできるようにする方式も提案されているが，すべてのハードウェアに対して直接アクセスできるようにすると，仮想マシンモニタ自身を動作させることが困難になってしまう．

本論文では，仮想マシンモニタを用いた透過的なネットワークブート方式を提案する．通常の PC と同等の動作環境を実現するために，ゲスト OS がローカルディスクへのアクセスと同様の方式でサーバのストレージにアクセスできるようにする．これにより，既存の OS をいっさい修正せずにネットワークブートすることが可能になる．また，通常の PC と同等の機能を実現するために，ストレージとネットワーク以外のデバイスへのアクセスは基本的にパススルーとする．これにより，OS はクライアント端末のハードウェアを直接アクセスすることができる．これらの 2 つの機能を両立させるために，仮想マシンモニタ内でホスト OS の機能に頼ることなく必要最小限の処理で，ストレージデバイスへのアクセスをネットワークを介したサーバへのアクセスに変換する．これにより，動作環境，機能，性能の 3 つすべての観点で透過的なネットワークブートシステムを実現する．

提案方式の実装は，準パススルー型の仮想マシンモニタである BitVisor[5)] をベースに行った．OS からストレージへのアクセスを透過的にするために，ATA デバイスへのアクセスを捕捉する機能を実装した．また，ストレージへのアクセスを最小限の変換でネットワークに転送するために，ATA デバイスへのアクセスを AoE（ATA over Ethernet）プロトコルに変換する機能と，ネットワークデバイスである Intel Pro 1000 経由で AoE プロトコルのパケットを送受信する機能を実装した．仮想化によるオーバヘッドを削減するために，ATA デバイスは完全には仮想化せず，必要最小限のアクセスのみを捕捉することで上記の機能を実現した．

実装したシステムの動作確認により，通常の PC 向けの Windows Vista，7 および Linux ディストリビューションの 1 つである Debian をいっさい改変することなく，そのままネットワークブートできることを確認した．また，USB デバイスなどをクライアント端末に接続して通常の PC と同様に利用できることも確認した．クライアントとサーバに同じ性能のマシンを用いて起動時間を測定した結果，提案方式はローカルディスクからブートする場合と比べて，サーバのキャッシュが有効な場合には全体で 14 秒短縮し，キャッシュを無効にした場合でも 24 秒の増加に抑えられることが分かった．

本論文の貢献は，以下のとおりである．

- ディスクへのアクセスをホスト OS の機能に頼ることなく最小限の機能でネットワークに転送する手法を示した．
- 実例として，ATA デバイスへのアクセスを最小限の機能で AoE プロトコルに変換してサーバのストレージへのアクセスする手法を示した．
- ディスクアクセスのみをネットワークに転送して，他のハードウェアはゲスト OS が直接ハードウェアにアクセスできる仮想マシンモニタを実現した．
- 性能評価を行い，実用的なオーバヘッドでネットワークブートが実現できることを明らかにした．
- 提案方式によって動作環境・機能・性能の 3 つの観点で透過的なネットワークブートシステムが実現可能であることを実証した．

以下，2 章で関連研究について述べる．3 章で提案方式の設計について説明し，4 章でその実装について述べる．5 章で性能評価を行い，6 章でまとめと今後の課題について述べる．



© 2011 Information Processing Society of Japan

## 2. 関連研究

従来のネットワークブート方式には，OS 方式，仮想マシンモニタ方式，ハードウェア方式などがある．また，ネットワークブート方式と並ぶシンクライアントの実現手法として，画面転送方式がある．以下，それぞれの方式について説明する．

### 2.1 OS 方式

UNIX 系の OS では，NFS[6] や AFS[7] などの分散ファイルシステムを用いてネットワークブートすることができる．また，最近の OS では，iSCSI[8] や AoE[9]，NBD（Network Block Device）[10] などブロックデバイスのレイヤでサーバのストレージにアクセスする仕組みがサポートされている．また，TransCom[11] や Ardence[3] では，独自のブロックデバイス・ドライバにより，サーバ上の仮想ディスクにアクセスして，ネットワークブートすることができる．

しかし，近年クライアント環境で多く用いられている Windows や Linux，FreeBSD などの OS の多くはローカルディスクからのブートを前提とした構成がなされており，ネットワークブートを行うためには専用のデバイスドライバやカーネルモジュールの導入，設定ファイルの作成などの複雑な設定が必要になる[2]．これらの設定は OS ごとに異なるほか，同じ OS でもディストリビューションごとに異なる場合がある．たとえば，Linux などオープンソース OS のディストリビューションは数百以上あるとされているが[12]，それらは必ずしも同じ設定でネットワークブートできるとは限らない．また，OS のアップデートやバージョンアップに際してシステムの構成が変わる可能性があり，ネットワークブートの設定に影響を与える可能性もあることから，ローカルディスクからのブートの場合と比べて追加で検証作業が必要になりシステム管理作業が面倒で複雑になる．

提案方式では，OS に依存せずにネットワークブートを実現できるため，ネットワークブートの設定とは独立してシステム管理作業を行うことができる．また，新しい OS の導入やアップデート，バージョンアップなどの構成変更をローカルディスクと同じ方式でできるほか，ソフトウェアアップデートなど頻繁に必要となるために自動化されていることが多い管理作業も，通常のローカルディスクからのブートを前提とした構成と同じやり方で行うことができる．

ディスクアクセスを逐一ネットワークに転送せず，ディスクイメージを事前にクライアント端末のハードディスクに配布することで，通常のローカルブートと同様の動作環境で動作する方式もあるが，この方式は事前に大量のデータをクライアントに配信する必要があるため，OS やアプリケーションのインストールやアップデートを適時行うことが難しくなる．

### 2.2 仮想マシンモニタ方式

Collective[1] や ISR[4] では，クライアント端末上に仮想マシンモニタを配置し，分散ファイルシステムを介してサーバ上の OS イメージをゲスト OS としてブートする．これらの方式では，ゲスト OS はネットワークブートであることを意識する必要がないものの，計算機全体が仮想化されてしまうため，ユーザやゲスト OS がクライアント端末のハードウェアのすべての機能を利用したり，効率的な資源管理や電源管理を行ったりすることが難しくなる．たとえば，ビデオ編集や 3D グラフィックスのための特殊な機能を持った PCI デバイスなどの利用が難しくなるほか，HDD や SSD の物理的特性を活用したファイルシステムのデータ配置を行ったり，ユーザの利用状況やアプリケーションの動作状況をふまえて適切なタイミングでデバイスの電源を ON/OFF するなどの電源管理が難しくなったりする．また，仮想化による様々なオーバヘッドがかかるほか，ゲスト OS を起動する前にホスト OS と仮想マシンモニタ自身を起動する時間がかかるため，システム全体としての起動時間が倍近くになる可能性がある．

最近の VMWare や Xen などでは，グラフィックスなど特定の PCI デバイスをパススルーとして，ゲスト OS が直接アクセスするができる．また，USB デバイスや CD/DVD デバイスなど指定した周辺機器をゲスト OS に接続することもできる．しかし，ホスト OS が動作するためにすべてのデバイスをパススルーにすることは難しい．たとえば，従来の仮想マシンモニタでネットワークブートを実現するためには，分散ファイルシステムにアクセスするための TCP/IP プロトコルスタックやファイルシステム，それらの基盤となるプロセス管理やメモリ管理など多数のホスト OS の機能が必要となる．ゲスト OS からハードウェアへ直接アクセスできるようにすると，ホスト OS をそのまま動作させることは難しく，相互に依存していることから部分的に機能を削ることは容易ではない．

本研究では，ディスクアクセスを最小限の機能でネットワークに転送する方式を実現することにより，ホスト OS の機能に依存することなくネットワークブートを可能にしつつ，ゲスト OS がハードウェアへ直接アクセスできる仮想マシンモニタを実現している．

### 2.3 ハードウェア方式

IDE-R[13] では，クライアント端末から IDE デバイスへのアクセスをサーバへとリダイレクトすることができる．IDE-R はリモート管理を目的としており，デバイスはフロッピーディスクと CD-ROM に限定されているほか，専用のハードウェアが必要となる．提案方式は，広く普及している汎用 PC とソフトウェアのみで実現しているため，高コストになりが



© 2011 Information Processing Society of Japan

ちな専用ハードウェアを使わずに低コストでネットワークブートシステムを実現できる．

BIOS などのファームウェアでディスクへのアクセスを転送する方式もあるが，この方式では OS がデバイスドライバを用いて直接デバイスにアクセスした場合には，アクセスを捕捉することが難しくなる．

### 2.4　画面転送方式

ネットワークブート方式と並ぶシンクライアントの実現方式として，画面転送方式がある[14]．この方式では，ソフトウェアはサーバで動作し，クライアント端末からネットワーク経由で遠隔操作する[15]．したがって，クライアント端末に接続できる機器が限定されたり，応答性が低下したりするなど，通常の PC と同等の機能や性能を実現することは難しい[16),17]．また，高価なサーバが必要となり，システム全体のコストが高くなる．

提案方式では，クライアント側に汎用 PC を用いて，そのハードウェア資源を最大限に有効活用することで，サーバ側に必要な機能を減らして，低コストなシステムを実現することができる．

## 3.　提案方式の設計

本章では，提案方式の設計について説明する．まず，仮想マシンモニタのアーキテクチャについて述べ，次にサーバソフトウェアで必要な機能について述べる．

### 3.1　仮想マシンモニタのアーキテクチャ

**図 1** に提案方式の設計を示す．提案方式では，クライアント端末上に仮想マシンモニタを配置し，サーバの OS イメージをネットワーク経由でブートする．透過的なネットワークブートを実現するために，仮想マシンモニタでゲスト OS からストレージへのアクセスを捕捉し，それをネットワーク経由でサーバへ転送する．図 1 では，OS における「ストレージへのアクセス」が，仮想マシンモニタ内の各モジュールやドライバおよびネットワークデバイスを経由して，「サーバへのアクセス」に変換されている様子を示している．

一方，機能面と性能面での透過性を得るために，ネットワークブートに使用するストレージおよびネットワーク以外のデバイスは基本的にパススルーとし，ゲスト OS に直接操作させる．図 1 では，「内蔵機器」や「周辺機器」へのアクセスがパススルーになっている．通常の PC クライアント端末上で動作する OS は元々1 つであるため，仮想マシンモニタ上でもゲスト OS は 1 つだけ動作させる．これにより，ゲスト OS 間のスケジューリングやリソース管理などを省いて仮想化のオーバヘッドが抑えられるほか，ゲスト OS が通常の PC と同様に内蔵機器や周辺機器にアクセスできるため，機能面や性能面での透過性が向上する．

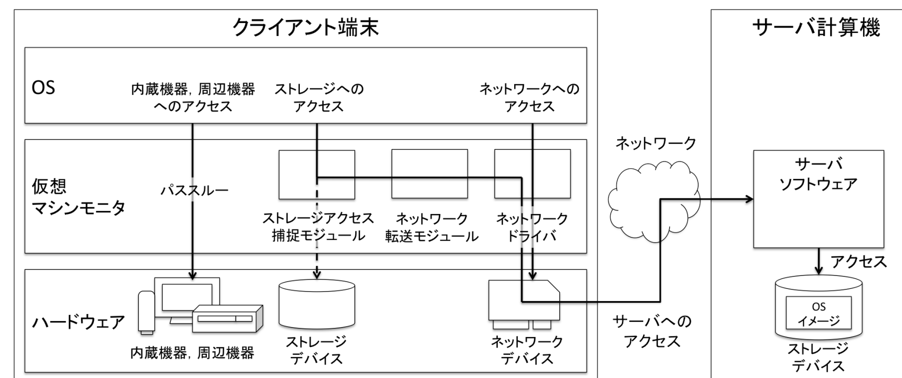


**図 1**　提案方式の設計
Fig. 1　Design of our system.

仮想マシンモニタは，図 1 に示すように，(1) ストレージアクセス捕捉モジュール，(2) ネットワーク転送モジュール，(3) ネットワークドライバの 3 つの構成要素を持つ．以下，それぞれの要素を説明する．

#### 3.1.1　ストレージアクセス捕捉モジュール

ストレージアクセス捕捉モジュールでは，ゲスト OS によるストレージへのアクセスを捕捉する．捕捉したアクセスの内容は，ネットワーク転送モジュールに渡してサーバへと転送する．サーバでのアクセスの結果が通知されると，本モジュールはその結果を解釈してデバイスの状態に反映させる．また，DMA による読み込みの場合には，読み込んだセクタ・データをゲスト OS のメモリに書き込む処理などデバイスの動作をエミュレーションする処理を行う．また，必要に応じて，処理の完了を通知するためにゲスト OS に対して割込みを発生させる．

ストレージアクセス捕捉モジュールでは，オーバヘッドの削減など透過性の向上を図るために，クライアント端末に存在する物理的なストレージデバイスを活用して最小限の仮想化のみを行う．たとえば，デバイスのレジスタへの読み書きなどは可能な限り実在のデバイスのレジスタを活用し，仮想マシンモニタに制御がわたる回数を削減する．また，実在のデバイスのインタフェースを踏襲することにより，ゲスト OS からは通常のローカルデバイスと同じようにアクセスできるようにする．このとき，実在のデバイスとしては，ATA や SCSI など標準的なインタフェースを持ったデバイスを用いることで，ほとんどの OS を



© 2011 Information Processing Society of Japan

ネットワークブートすることが可能になる．

#### 3.1.2　ネットワーク転送モジュール

ネットワーク転送モジュールでは，ストレージアクセス捕捉モジュールで捕捉した内容をサーバへのアクセスに変換し，ネットワークドライバを用いてサーバへ転送する．アクセスが完了すると，その結果をストレージアクセス捕捉モジュールに返す．

ネットワーク転送モジュールでは，ネットワーク経由でデータをやりとりする際の変換のオーバヘッドを削減するために，NFS などのファイルシステムレベルではなく，AoE や iSCSI などのストレージレベルの通信プロトコルを用いる．また，プロトコルを選択する際は，ストレージアクセス捕捉モジュールでアクセスを捕捉するデバイスとなるべくインタフェースが近いものを選択する．たとえば，ATA デバイスでは AoE，SCSI デバイスでは iSCSI など類似のインタフェースを持つことが望ましい．

これらのプロトコルでは，ネットワーク送受信の際に特定のフォーマットのパケットを用いるが，これらのパケットはデバイスレベルでのアクセスと 1 対 1 に対応しているとは限らない．たとえば，デバイスのレジスタへのアクセスを逐一サーバに送るのではなく，「セクタの読み込み」といったまとまった単位で行う．そこで，ネットワーク転送モジュールでは，ストレージアクセス捕捉モジュールで捕捉したアクセスをバッファリングして，読み込みコマンドが発行されたときなど適切なタイミングでパケットへの変換を行う．

また，ネットワーク転送モジュールは，ストレージレベルのプロトコルのプロトコルスタックを持ち，データの転送が確実に行われることを保証する．たとえば，タイマによりパケットロスを検出して再送要求を行ったり，重複パケットやエラーパケットを検出して破棄したりするなど，信頼性のある通信を実現する．

#### 3.1.3　ネットワークドライバ

ネットワークドライバは，仮想マシンモニタがストレージレベルの通信プロトコルのパケットを送受信するために用いる．ゲスト OS がネットワークデバイスの初期化を始めるまでは，仮想マシンモニタがネットワークデバイスを完全に管理することができるため，ネットワークドライバは必要最小限の機能を実装したドライバを用いてパケット送受信を行う．

ゲスト OS がネットワークデバイスの初期化を始めた後は，ゲスト OS と仮想マシンモニタの両方がネットワーク送受信をできるようにするために，ネットワークデバイスへのアクセスを部分的に仮想化してパケット送受信を多重化し，ストレージアクセスのネットワーク転送が途切れないようにする．

### 3.2　サーバソフトウェアの機能

サーバソフトウェアでは，クライアント端末から転送されるアクセスの内容に従って，図 1 の右側に示すように，実際の OS イメージが格納されているストレージデバイスに対してアクセスを行う．

3.1.2 項で述べたように，クライアント側のネットワーク転送モジュールでは，変換のオーバヘッドを削減するために，AoE や iSCSI などブロックレベルでの転送プロトコルを用いることを想定している．したがって，サーバソフトウェアは，これらのプロトコルをサポートするものが必要となる．また，必要に応じて，OS イメージのコピーオンライトなどの機能を用いることにより，多数のクライアントの OS イメージを効率良く持つことができる[1)]．さらに，サーバ側でプロトコル変換処理やネットワーク転送処理などを行うことで，異なるプロトコルの相互接続を行ったり，NFS などの分散ファイルシステムやインターネット経由でのアクセスなどを実現したりすることもできる．

## 4.　提案方式の実装

本章では，提案方式の実装について述べる．まず，全体の概要を述べ，続いて，クライアント側，サーバ側それぞれの実装の詳細について述べる．

### 4.1　概　　要

提案方式の実装は，準パススルー型アーキテクチャを持つ仮想マシンモニタである BitVisor[5)] をベースにした．準パススルー型アーキテクチャとは，ゲスト OS からデバイスへのアクセスを基本的にはパススルーしつつ，最低限必要な I/O だけを仮想マシンモニタで捕捉する仕組みである．ゲスト OS は 1 つだけ動作させ，デバイスの大部分をゲスト OS に管理させて仮想マシンモニタを簡略化している．

本実装では，BitVisor の内部で動作するモジュールとして，3 章で述べた 3 つの構成要素を実装した．(1) **ストレージアクセス捕捉モジュール**（4.2 節）では，ストレージデバイスとして ATA デバイスを対象とし，BitVisor が持つデバイス I/O の捕捉機能を利用して，指定した ATA デバイスのレジスタへのアクセスを捕捉した．(2) **ネットワーク転送モジュール**（4.3 節）では，AoE プロトコルを用いてストレージにアクセスする機能および ATA デバイスへのアクセスとの相互変換を行う機能を実装した．(3) **ネットワークドライバ**（4.4 節）では，ネットワークデバイスである Intel Pro 1000 を対象として，ゲスト OS が起動する前にネットワーク送受信を行うためのドライバを実装した．

さらに，BitVisor 内で動作する補助モジュールとして，起動時に AoE サーバを検索して



© 2011 Information Processing Society of Japan

コマンドブロックレジスタ

| レジスタ名(リード) | レジスタ名(ライト) |
|---|---|
| Data | Data |
| Error | Features |
| Sector Count | Sector Count |
| LBA Low | LBA Low |
| LBA Middle | LBA Middle |
| LBA High | LBA High |
| Device | Device |
| Status | Command |

コントロールブロックレジスタ

| レジスタ名(リード) | レジスタ名(ライト) |
|---|---|
| Alternate Status | Device Control |

DMAバスマスタレジスタ

| レジスタ名 |
|---|
| BM Command |
| BM Status |
| BM Descriptor Pointer |

**図 2**　ATA レジスタ
Fig. 2　ATA registers.

クライアント端末の ATA デバイスとの対応を管理する**デバイス管理モジュール**（4.5.1 項）と，仮想マシンモニタ自身の機能を補うための**割込み管理モジュール**（4.5.2 項）を実装した．また，AoE サーバとしては，vblade [18] をベースとして使用し，ネットワークブートに必要な機能の補うための実装（4.6 節）を追加した．

以下では，それぞれの実装について詳しく述べる．

### 4.2　ストレージアクセス捕捉モジュールの実装

本実装では ATA デバイスを対象として実装を行った．本節では，まず ATA 規格の概要について述べ，次に ATA レジスタの仮想化について述べる．続いて本モジュールの動作フローについて述べ，最後に割込み処理について述べる．

#### 4.2.1　ATA 規格

ATA 規格では，ATA ホストコントローラに複数の ATA デバイスが接続される構造を持つ．OS は，ATA ホストコントローラが持つ ATA レジスタと呼ばれるレジスタ群を I/O 命令で読み書きして，ATA デバイスを操作する．このレジスタ群は，**図 2** のようにコマンドブロックレジスタとコントロールブロックレジスタ，DMA バスマスタレジスタからなる．

ATA デバイスの操作は，ATA コマンドプロトコルと呼ばれる手順に従って，ATA レジスタを読み書きすることで行う．基本的な ATA コマンドプロトコルには，Non data，PIO data-in，PIO data-out，DMA の 4 つがある．ここでは OS をネットワークブートするにあたって重要な PIO data-in および DMA コマンドプロトコルについて説明する．

PIO data-in コマンドプロトコルは，ATA デバイスからデータを読み込む際に OS が I/O を繰り返す方法である．まず，Sector Count，LBA Low，LBA Middle，LBA High の各レジスタに対して，読み込むデータの先頭セクタ番号，セクタ数を指定する．次に，Command レジスタにコマンド番号を書き込む．これにより，ATA デバイスに対してコマンドが発行され，データの読み込み準備が開始される．準備が完了するまでの間，Status レジスタには準備中を示す値が入る．

準備が完了すると，ATA デバイスは Status レジスタの値に準備完了を示す値を設定し，必要に応じて割込みを発生する．OS は，割込みによる通知もしくは Status レジスタの値を読むことで，読み込み準備完了を検知する．読み込み準備完了を確認した OS は，Data レジスタの値を連続読み込みすることで，1 セクタ分のデータを読み込む．複数のセクタを読み込む場合は，Status レジスタの値を確認して，ATA デバイスの準備ができ次第，次の連続読み込みを行う．

一方，DMA コマンドプロトコルでは，DMA バスマスタと呼ばれるデバイスが自動的に指定したアドレスへデータ転送を行う．DMA コマンドプロトコルでは，ATA コマンド発行までの処理は PIO data-in と同様だが，その後バスマスタの Command レジスタの開始ビットをセットすることで，DMA バスマスタによる転送が開始される．転送の完了は割込みで通知され，Status レジスタの値で結果を確認する．

#### 4.2.2　ATA レジスタの仮想化

ストレージアクセス捕捉モジュールでは，ローカルの ATA デバイスを活用して，必要最小限の仮想化で ATA コマンドを捕捉する．ATA レジスタ群のうち，LBA と Sector Count は値の読み書きでデバイスの状態が変わらないため，本モジュールではこのレジスタへのアクセスは捕捉しない．一方，Command，Status，Error，Device Control，Data，BM Command の各レジスタは，値の読み書きによってデバイスの状態が変わるため，本モジュールでアクセスを捕捉して，仮想レジスタへの読み書きへと変換する．

Command レジスタは，ATA コマンドの発行に使われる．レジスタに値が書き込まれた時点でコマンドが発行されるため，本モジュールでアクセスを捕捉し，書き込まれた値を仮想レジスタに格納する．一方，Status レジスタは，ATA デバイスの動作状況を取得するために使われる．したがって，本モジュール内部の状態に合わせて準備中もしくは準備完了な


© 2011 Information Processing Society of Japan

どの値を返す必要がある．また，Error レジスタは，ATA デバイスにエラーが発生したときの状況を知らせるために使われる．そこで，これらのレジスタへのアクセスを捕捉して，仮想レジスタの値を返すようにする．

Device Control レジスタの値は，ATA コマンド完了時に割込みを発生するか否かを決定する．しかし，Device Control レジスタは書き込み専用レジスタである．そこで Device Control レジスタの値を知るため，ゲスト OS によるこのレジスタへの書き込みを捕捉し，本モジュール内の仮想レジスタに値を保持する．

PIO data-in コマンドプロトコル，PIO data-out コマンドプロトコルでは，OS は Data レジスタに繰り返しアクセスすることで，データを読み書きする．そこで Data レジスタへのアクセスを捕捉し，サーバから転送したデータを格納したバッファに対して読み書きを行わせる．一方，DMA コマンドプロトコルでは，BM Command レジスタの開始ビットを立てることで，DMA バスマスタによる転送が開始される．そこで，このレジスタへのアクセスも捕捉し，DMA によるデータの転送をエミュレートする．

#### 4.2.3　動作フロー

ストレージアクセス捕捉モジュールに制御がわたるタイミングには，(1) ATA コマンド発行，(2) サーバへのアクセス完了，(3) ゲスト OS によるデータ取得の 3 つがある．それぞれのタイミングで始まる動作フローは ATA コマンドプロトコルの内容に応じて異なるが，大まかに PIO 系と DMA 系の 2 つの ATA コマンドプロトコルに分けて説明する．

ATA コマンド発行を契機とする動作フローは，ゲスト OS が Command レジスタに値を書き込んだ時点で始まる．本モジュールはその時点の仮想・実レジスタの値を取得し，先頭セクタ番号やセクタ数など必要な情報を抽出する．同時に，仮想 Status レジスタに準備中を表す適切な値を設定する．PIO 系コマンドプロトコルの場合は，この時点で抽出した情報を構造体に格納し，ネットワーク転送モジュールに渡す．

サーバへのアクセス完了を契機とする動作フローは，ネットワーク転送モジュールでサーバへのアクセスが完了したときに始まる．本モジュールはアクセス結果を構造体として受け取り，その内容を解釈して各レジスタの値に反映させる．DMA 系コマンドプロトコルでの読み込みの場合は，DMA バスマスタレジスタの値を読んでゲスト OS が指定したバッファのアドレスを取得し，サーバから取得したデータの転送を行う．

ゲスト OS によるデータ取得を契機とする動作フローは，PIO 系と DMA 系で内容が異なる．PIO 系の場合は，ゲスト OS が Data レジスタにアクセスした時点で始まり，仮想 Data レジスタを介してデータの読み書きを行わせる．DMA 系の場合は，DMA バスマスタレジスタの開始ビットをセットした時点で始まり，DMA バスマスタレジスタの値を読むことで，ゲスト OS が指定したバッファのアドレスを取得する．アクセスが書き込みの場合は，この時点でバッファのデータを取得し，ATA レジスタから抽出した情報と合わせて，ネットワーク転送モジュールに渡す．読み込みの場合は，この時点では単に ATA レジスタから抽出した情報のみを渡し，サーバへのアクセスが完了した時点でバッファにデータを転送する．

#### 4.2.4　割　込　み

ATA コマンドの完了は，ATA デバイスからの割込みによってゲスト OS に通知される．これをエミュレートするために，4.5.2 項で述べる割込み管理モジュールの機能を用いて，ATA デバイスに割当てられている割込み番号を取得し，必要に応じてその番号に対して INT 命令で仮想マシンモニタから割込みを発生させる．

### 4.3　ネットワーク転送モジュールの実装

本実装では，転送プロトコルに AoE を用いた．本節では，まず AoE プロトコルの概要について述べ，次に AoE プロトコルスタックの実装について述べる．

#### 4.3.1　AoE プロトコル

AoE プロトコルは，ATA コマンドを Ethernet で転送するためのシンプルな通信プロトコルである．AoE プロトコルのパケットは，共通ヘッダの後にパケットの種類ごとに異なるヘッダが続く構造になっている．各ヘッダの内容を**図 3** に示す．共通ヘッダは，Ethernet ヘッダの後にバージョン（Ver），フラグ（Flags），エラー番号（Error）が続き，さらにアクセス対象の ATA デバイスのメジャー番号（Major）とマイナー番号（Minor），コマンド番号（Command）が続く．

AoE プロトコルでは，要求パケット 1 つに対して応答パケットが 1 つ返る．その際，要求パケットと応答パケットの対応関係を保つために，共通ヘッダの Tag フィールドを用いる．サーバは，受信した要求パケットの Tag フィールドを，応答パケットの Tag フィールドにコピーして送信する．1 つのパケットで送受信できるセクタ数は Ethernet のフレームサイズで制限され，1500 バイトのフレームの場合は 2 セクタである．

AoE プロトコルのパケットには，ATA コマンドパケットとコンフィグパケットの 2 種類がある．ATA コマンドパケットは，ATA コマンドをネットワーク転送するパケットであり，ATA レジスタをそのまま格納するようなフィールドを持つ．また，転送データが存在する場合は，ペイロードとしてセクタ単位でデータが続く．コンフィグパケットは，サーバやそのサーバが保持する ATA デバイスの情報を取得，設定するためのパケットである．



© 2011 Information Processing Society of Japan

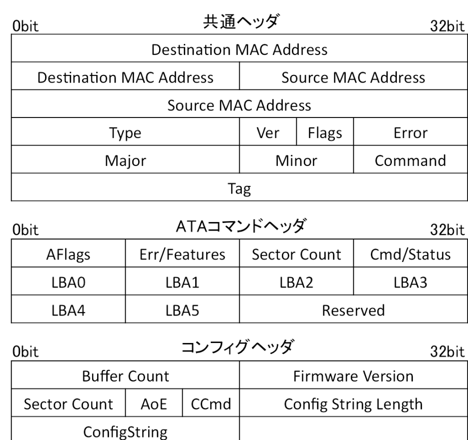


**図 3**　AoE パケットのヘッダ
Fig. 3　AoE headers.

#### 4.3.2　AoE プロトコルスタック

AoE プロトコルスタックは，ATA コマンドから AoE パケットへの変換および AoE パケットの送受信を行う．

ストレージアクセス捕捉モジュールから受け取る ATA コマンドは，コマンド単位で構造体として渡されるので，それを AoE パケットへと変換する．ATA コマンドと AoE パケットは，ほぼ同レベルの情報を持つため，基本的には単純に形式を変換するだけでよい．ただし，AoE パケットでは 1 度に転送できる最大セクタ数に制限があるため，必要に応じて 1 つの ATA コマンドから複数の AoE パケットを生成する．また，発行された ATA コマンドと AoE パケットとの対応関係を管理して，すべての応答パケットを受信した時点でストレージアクセス捕捉モジュールに通知する．

AoE プロトコルでは再送機能は定義されていないが，OS のネットワークブート時には大量の AoE パケットが送受信されるため，パケットロスに対応するため再送機能の実装が必要となる．本実装では，再送管理のために，送信した要求パケットとすでに返信された応答パケットの対応関係をテーブルで保持する．また，要求パケットを送信する際に再送タイマを動作させ，一定時間経過後に未受信の応答パケットに対応する要求パケットを再送する．応答パケットを重複して受信した場合は，不要なパケットを廃棄する．

上記のような，ATA コマンドと AoE パケットとの対応や再送の管理のために，本実装では AoE パケットの共通ヘッダの Tag フィールドを用いた．Tag フィールドは 2 つの 16 ビットの値に分割して，(1) **ATA コマンド番号**および (2) **AoE パケット番号**の 2 つの値を格納する．ATA コマンド番号は，ATA コマンドを発行するたびにカウントアップするシーケンス番号で，受信した応答パケットがどの要求パケットに対応するかを判断するために用いる．AoE パケット番号は，1 つの ATA コマンドから複数の要求パケットを生成した場合に，各パケットを区別するために用いる．

### 4.4　ネットワークドライバの実装

本実装では，ネットワークデバイスとして Intel Pro 1000 を対象とし，BitVisor がすでに持っている準パススルー型ネットワークドライバを流用した．このドライバでは，ゲスト OS のドライバにデバイスの管理の多くをまかせつつ，仮想マシンモニタによるパケットの監視と多重化が可能な構造になっている．したがって，この多重化機能を利用して，ゲスト OS が稼働中に仮想マシンモニタから AoE パケットを送受信することが可能である．

しかし，BitVisor のドライバはゲスト OS が NIC（Network Interface Card）を初期化して稼働状態にすることを前提としており，ゲスト OS の起動時など NIC の初期化を行う前の状態では機能しない．そこで，本実装では，仮想マシンモニタの起動時に，独自に最低限の NIC 初期化処理を行う実装を追加した．これにより，(1) ゲスト OS のドライバが NIC を初期化する前，および (2) NIC を初期化した後のいずれの場合でも，仮想マシンモニタによるパケット送受信が可能となる．

ここで，ゲスト OS が NIC の初期化や停止を行った場合，その処理をそのまま NIC に反映すると，仮想マシンモニタからもパケット送受信が不可能になる．したがって，仮想マシンモニタからのパケット送受信を継続するためには，それらの初期化や停止処理を捕捉して，NIC からの送受信が中断されないようにする必要がある．さらに，このときゲスト OS に対して初期化や停止処理が正しく反映されたように見せかけるために，NIC の状態を一時的に仮想化する必要がある．

現在の実装では，この NIC の初期化・停止処理を一時的に仮想化する部分の実装が不十分であり，ゲスト OS が NIC を初期化した後に NIC にアクセスできないという制限が残っている．したがって，現段階では，ゲスト OS から NIC を隠蔽して初期化を行わせないことでネットワークブートを継続できるようにしてあり，ゲスト OS は仮想マシンモニタが使用している NIC にアクセスすることができない．しかし，将来的には一時的な仮想化を適切に実装することで，ゲスト OS からの NIC アクセスも可能になると考えられる．


© 2011 Information Processing Society of Japan

### 4.5　補助モジュールの実装

補助モジュールとして，デバイス管理モジュールと割込み管理モジュールの 2 つを実装した．本節では，それぞれの実装について述べる．

#### 4.5.1　デバイス管理モジュールの実装

デバイス管理モジュールでは，ローカルの ATA ホストコントローラと AoE サーバが提供する ATA デバイスとの対応関係を管理する．

ローカルの ATA ホストコントローラの検出は，BitVisor の PCI デバイス検出機能を用いて行う．AoE サーバの ATA デバイスの検出は，AoE プロトコルのコンフィグパケットをブロードキャストして，ネットワーク上のすべての ATA デバイスの情報を取得することで行う．ローカル ATA デバイスとネットワーク上の ATA デバイスの対応関係は，BitVisor コンパイル時に静的に設定することとした．

なお，既存の BitVisor では，すべての ATA デバイスと ATAPI デバイスを検出して I/O アクセスを捕捉する仕様になっている．そこで，CD/DVD ドライブなどの ATAPI デバイスはパススルーとして透過性を上げるために，デバイスの種類を判別して I/O 捕捉の有無を切り替える機能を実装した．

#### 4.5.2　割込み管理モジュールの実装

OS に対する割込みは，割込みベクタ番号と呼ばれる番号によって管理される．OS は，この割込みベクタ番号と各デバイスの対応付けを行っており，割込み発生時にベクタ番号を調べることで，対応するデバイスを認識している．

提案方式では，ATA コマンドの完了をゲスト OS に通知する際，ATA デバイスからの割込みを発生させる必要がある．また，NIC によるパケットの送受信の際に，仮想マシンモニタで NIC からの割込みを認識する必要がある．したがって，これらのデバイスに対応する割込みベクタ番号を把握する必要がある．

既存の BitVisor では，特定の割込みベクタ番号に対して，割込みを捕捉する機能や割込みを発生させる機能は実装されている．しかし，割込みベクタ番号とデバイスの対応関係を取得する機構が実装されていない．そこで，その対応関係を取得する機能を割込み管理モジュールとして実装した．

デバイスと割込みベクタ番号の対応関係は，割込みコントローラと呼ばれるチップが保持している．この対応関係の設定は，BIOS によってデフォルトの設定が行われるほか，OS が起動時に新しい値を設定することがある．しかし，割込みコントローラは仕様上，設定された対応関係を後から読み出すことが困難である．そこで，ゲスト OS 起動前は BIOS の仕様書に記載されたデフォルト値を静的に保持して使用する．また，ゲスト OS 起動後は，割込みコントローラへのアクセスを本モジュールで捕捉して，ゲスト OS が設定する対応関係を把握する．

また，4.4 節末尾で述べたように，ゲスト OS は NIC を認識しないため，NIC の割込み管理においても BitVisor 独自で行う必要がある．そのため，通常ゲスト OS が利用しない 0xFF 番割込みを BitVisor で NIC に割り当てる実装を行った．ただ，実際にゲスト OS に 0xFF 番割込みが利用されているのを検知されてはならないので，割込み管理モジュールでゲスト OS から見て 0xFF 番割込みが未使用に見えるような仮想化を部分的に行った．ゆえに，0xFF 番割込みは実際 BitVisor が利用しているため，ゲスト OS からは利用不可能であることが制約となっている．ただし，4.4 節末尾で述べた NIC の実装が完了し，ゲスト OS が NIC を認識するようになれば，BitVisor が NIC の割込みを独自に管理する必要がなくなるため，割込みを仮想化することによる制約は解消する．

### 4.6　AoE サーバの改良

AoE サーバには，Linux で動作する vblade を用いた．vblade は，ブロックデバイスやファイルを ATA デバイスとして指定することができる．したがって，OS イメージを格納したファイルなどを指定することにより，その OS イメージからネットワークブートできる．

しかし，vblade では，本来ディスクが標準でサポートすべき ATA コマンドが十分に実装されておらず，ネットワークブートができなかったため，一部 ATA コマンドを追加実装した．たとえば，ATA デバイスの情報を取得する IDENTIFY DEVICE コマンドに対して，返答する ATA デバイス情報が不十分な内容となっていた．そこで，この情報に対して DMA 対応の有無など OS をブートするために必要な情報を追加する実装を行った．また，PIO，DMA を切り替えるコマンドが実装されていなかったので追加実装を行った．

### 4.7　BitVisor の起動

BitVisor 自体もサーバ上で集中管理できるようにするため，サーバ上から BitVisor をネットワークブートできる環境を構築した．BitVisor 自体のネットワークブートは，gPXE[19] と呼ばれるブートローダを用いて実現した．gPXE は，HTTP や FTP，TFTP などといったプロトコルを介して，サーバ上の BitVisor のイメージファイルをクライアントに読み込んで起動することができる．また，gPXE 自体は，CD や USB デバイス，ローカルディスクから起動することができるうえ，PXE を用いてネットワークブートすることもできる．

BitVisor はネットワークブートすると自身を初期化した後，BitVisor 上で BIOS を動作させる．BIOS は OS の起動処理を開始するが，BitVisor が起動しているため，すべての


© 2011 Information Processing Society of Japan

ディスクアクセスがネットワーク転送され，サーバ上のイメージから OS が起動する．

### 4.8 透過性に関する制約

BitVisor は CPU を仮想化しているため，CPU 仮想化支援機能などの特殊な CPU 拡張機能に関しては，現在 BitVisor の仮想化が対応していないため利用できない．ただし，仮想化をネストする技術[20] も存在するため，この技術を BitVisor に適用することで，将来 CPU 仮想化支援機能に関しては透過的にできる可能性がある．メモリにおいては，BitVisor がメモリ領域の一部を利用するため，ゲスト OS は実メモリ領域をすべて利用することはできない．

ネットワークデバイスに関しても，4.4 節で述べたように，実装上の制約から，ディスクアクセスをネットワーク転送するための NIC は隠蔽されているため，ゲスト OS からは利用できない．また，4.5.2 項で述べたように，0xFF 番割込みは現段階の実装ではゲスト OS が利用することはできない．

## 5. 実　　験

本章では，提案方式の透過性を検証するために，3 章で述べた方式で実装したシステムを用いて動作確認と性能評価の実験を行った．以下では，まず動作確認実験の結果について述べる．次に，OS の起動時間とディスクアクセス速度の測定結果について述べる．また，それらの測定結果に影響を与えたと考えられるサーバ OS のキャッシュの効果について述べる．そして，アプリケーションベンチマークの評価結果について述べる．最後に提案方式と既存の仮想マシンモニタ方式との性能比較について述べる．

本実験に用いた，サーバ・クライアントマシンを**表 1** に示す．本実験では対等な比較を行うために，サーバ・クライアントのマシンを同一の性能にした．サーバ・クライアント間は，スイッチングハブ FXG-08EMB を 1 台挟み，1 Gbps のイーサネットで接続した．サーバマシンでは AoE サーバとして Ubuntu 10.04 上の vblade 2.0 を用いた．クライアントマシンでは，VMM として BitVisor 1.1 を用いた．また，各性能評価実験ではクライアント OS として Windows 7（Business Edition）を用いた．

**表 1** 実験マシン
Table 1 Experimental environment.

| | |
|---|---|
| CPU | Intel Core 2 Duo P9500 @ 2.53 GHz |
| メモリ | PC2-6400 2 GB |
| ディスク | Western Digital Scorpio 80 GB 1.5 Gb/s 2.5 MB Cache |
| NIC | Intel 82567LM GbE Controller |

提案方式の動作には，クライアントマシンが CPU 仮想化支援機能（Intel VT または AMD-V），ATA ホストコントローラ，ネットワークデバイスの Intel PRO/1000 を搭載している必要がある．また，ATA ホストコントローラにはディスクが接続されている必要はないためディスクレスでも利用可能である．BitVisor 自体は，USB やネットワークから起動できる．サーバマシンには，AoE サーバが動作する範囲であれば任意のマシンを利用できる．

### 5.1 動作確認

提案方式が動作環境の点で透過的であることを示すために，既存の通常の PC 向け OS をいっさい改変せずにネットワークブートできることを確認する実験を行った．実験では，Linux の標準的なディストリビューションである Debian 5.0（Lenny）と，通常の PC 向け OS として一般的な Windows Vista（Business Edition），Windows 7（Business Edition）を使用した．OS イメージの作成には，PC エミュレータである QEMU を用い，通常の PC と同様の手順でイメージファイルにインストールを行った．作成したイメージファイルを vblade に指定し，クライアント端末から提案方式でネットワークブートした．提案方式で CD から OS をサーバ上のディスクにインストールすることは可能であるが，BitVisor の制約で一部機種に限られるうえ，本実験に用いた PC では利用できなかったため，QEMU を利用した．実験の結果，Debian 5.0（Lenny），Windows Vista（Business Edition），Windows 7（Business Edition）ともに，OS イメージそのものはいっさい改変せずに，提案方式によってネットワークブートできることを確認した．

また，提案方式が機能面でも透過的であることを示すために，ネットワークブートした OS からクライアント端末の周辺機器・内蔵機器が利用できることを確認した．具体的には，USB2.0/3.0 で接続されたフラッシュメモリや外付けハードディスク，端末に内蔵されている CD/DVD/Blu-ray ドライブからデータの読み書きができることを確認した．また，OS からグラフィックボードの機能を認識でき，デュアルディスプレイが可能なことも確認した．

ただし，4.8 節で述べたとおり，BitVisor が未対応であるため，起動した OS 上で実行される仮想マシンモニタは CPU 仮想化支援機能を利用できない．また，ネットワークデバイスに関しても 4.4 節で述べたように実装上の制約があり，現段階の実装では 1 個は OS からは利用できない．

### 5.2 起動時間

提案方式において OS の起動に要する時間を測定した．測定は，ブートローダである GRUB


© 2011 Information Processing Society of Japan

の選択画面で OS を選択してから，ログイン画面が表示されるまでにかかる時間をストップウォッチを用いて行った．また，BitVisor 自体の起動時間も測定した．比較のために，提案方式に加え，OS をローカルディスクから起動した場合も測定した．さらに，提案方式において，サーバ OS のキャッシュを有効にした場合と，無効にした場合でも測定した．これは，サーバ OS のキャッシュによる影響を取り除いて，ローカルディスクからの起動時間と，ネットワーク上のディスクからの起動時間を対等に比較するためである．クライアント OS がローカルディスクから起動した場合，クライアント OS 上でキャッシュミスしたディスクアクセスはディスクに対して行われる．それに対して，クライアント OS がネットワークブートした場合は．クライアント OS でキャッシュミスしたディスクアクセスでも，サーバ OS のキャッシュにヒットして，実質ディスクではなくメモリに対してのアクセスになってしまう．キャッシュを無効にする際は，イメージファイルの open() 時に O_DIRECT を用いるオプションを指定して vblade を起動し，サーバ OS のキャッシュを介さずにイメージファイルにアクセスさせた．Linux のページキャッシュでは，1 度アクセスが行われた領域は，キャッシュにデータが残り，同じ領域に対する 2 度目以降のアクセスがキャッシュヒットしてしまう．O_DIRECT を用いると，サーバ OS のキャッシュを回避することができ，同じ領域に幾度アクセスされてもキャッシュヒットが生じない．なお，vblade 自体にキャッシュは実装されていない．

測定結果を**図 4** に示す．提案方式において，キャッシュを有効にしてネットワークブートした場合，OS の起動時間は，ローカルディスクから起動した場合に比べて 14 秒短縮され，VMM 自体の起動時間を合わせても 8 秒短縮される結果となった．これは，ネットワーク転送したディスクアクセスが，サーバ OS のキャッシュであるメモリにヒットすることで，ローカルディスクへのアクセスよりも高速化したためと考えられる．なお，BitVisor 上でローカルディスクから起動した場合は，通常にローカルディスクから起動した場合と，OS の起動時間はほぼ同様であった．一方，キャッシュを無効にしてネットワークブートした場合，OS の起動時間は，ローカルディスクから起動した場合に比べて 24 秒増加した．VMM 自体の起動時間を合わせると，30 秒程度増加した．これは，サーバ OS のキャッシュは無効にしてあるため，提案方式によってディスクアクセスをネットワーク転送することで生じた純粋なオーバヘッドが現れていると考えられる．ただし，このオーバヘッドは，本実験に用いた既存のサーバアプリケーションである vblade の実装に因るところが大きく，サーバの実装の改善もしくは，より高速なサーバを用意することで抑えられると考えられる．このオーバヘッドの分析は，5.5 節で詳しく述べる．また，サーバ OS の起動時におけるキャッシュの影響については，5.4.1 項で詳しく述べる．

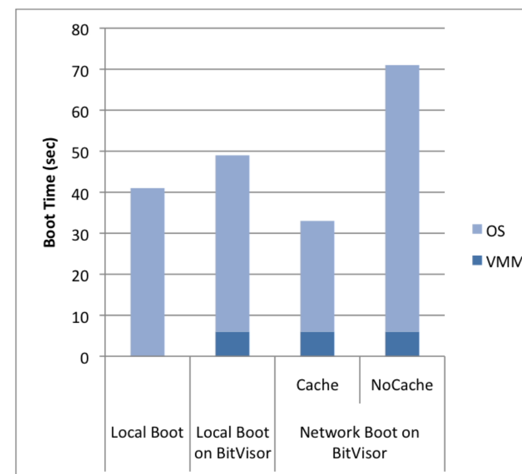


**図 4**　OS 起動時間
Fig. 4　OS boot time.

### 5.3　ディスクアクセス速度

提案方式におけるディスクアクセスのオーバヘッドを測定するために，ディスクアクセスのスループットを計測するベンチマークを行った．実験にはディスクベンチマークソフトである Crystal Disk Mark [21] を利用し，OS 上からディスクへのシーケンシャルアクセスおよびランダムアクセスのスループットを測定した．比較のため，提案方式に加え，OS をローカルディスクから起動した場合，BitVisor 上で OS をローカルディスクから起動した場合でも測定した．さらに，起動時間の測定と同様，提案方式においても，サーバ OS のキャッシュを有効にした場合と，無効にした場合で測定した．

測定結果を**図 5** に示す．各グラフは上から順にそれぞれ，レコードサイズ 1024 K バイトのシーケンシャルアクセス，レコードサイズ 512 K バイト，4 K バイトのランダムアクセスのスループットを示す．ローカルディスクから起動した場合はシーケンシャルリードが 45.8 MB/sec，シーケンシャルライトが 45.2 MB/sec，レコードサイズ 512 K のランダムリードが 24.8 MB/sec，ランダムライトが 30.8 MB/sec となった．一方，サーバ OS のキャッシュを有効にした状態で OS をネットワークブートした場合は，ローカルディスクから起動した場合に比べて，シーケンシャルリードで 26.1%，シーケンシャルライトで 8.2%ス


© 2011 Information Processing Society of Japan

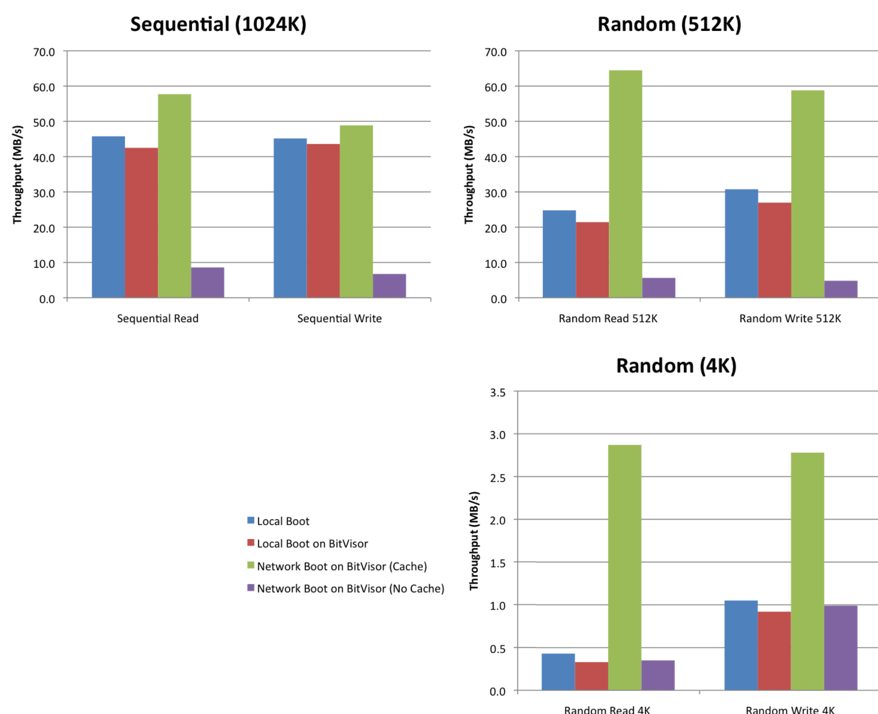


**図 5**　ディスクアクセス速度
Fig. 5　Disk access throughput.

ループットが向上した．また，レコードサイズ 4 K バイトの場合，ランダムリードで 6.7 倍，ランダムライトで 2.6 倍スループットまで向上した．これは，OS が発行したディスクアクセスがサーバ上のキャッシュにヒットしたため性能が向上したためと考えられる．リードよりライトのスループットが向上していない原因は，5.4.2 項で分析する．なお，BitVisor 上でローカルディスクから起動した場合は，通常のローカルディスクから起動した場合と，測定結果はほぼ同様であった．

一方，サーバ OS のキャッシュを無効にした状態で，OS をネットワークブートした場合は，シーケンシャルリードでは，ローカルディスクから起動した場合の 18.8%，シーケンシャルライトでは 14.9%にまでスループットが低下した．また，ランダムアクセスにおいても，レコードサイズ 512 K の場合，ランダムリードでは，ローカルディスクから起動した場合の 22.7%，ランダムライトで 15.7%にまでスループットが低下した．一方，レコードサイズ 4 K バイトのランダムアクセスに関しては，リード・ライトともに，ローカルディスクから起動した場合と同等であった．レコードサイズ 1024 K，512 K のディスクアクセスでスループットが著しく低下した原因は，5.5 節で詳しく述べる．

実験結果より，提案方式のディスクアクセス性能における透過性は，レコードサイズによって異なる可能性があるといえる．4 K バイト程度の小さなアクセスでは，キャッシュヒットの有無にかかわらず，OS をローカルディスクから起動した場合と同等以上であるため，ページのスワップイン・アウトやファイルなどへの小規模な読み書きなどにおいては，提案方式は透過的であるといえる．一方，巨大なファイルコピーなどでキャッシュミスが大量に発生した場合においては，透過性が制限される．ただし，5.5 節で詳しく述べるが，これはサーバアプリケーションの vblade の実装上の問題であり，サーバの実装を改善することで透過性が向上する可能性がある．また，5.6 節で示すアプリケーションベンチマークでは，ディスクアクセスが大量に発生するユースケースにおいても，OS をローカルディスクから起動した場合に比べ，ネットワークブートした場合の評価は同等以上であるため，通常のユースケースではキャッシュミスによる透過性の低下はそれほど大きくないといえる．

### 5.4　キャッシュの効果

提案方式において，キャッシュの効果を確認する補足実験を行った．以下では，まず，OS 起動時におけるキャッシュの効果について述べる．次に，集中的なディスクアクセス時のキャッシュの効果として，5.3 節でベンチマークを実行した際におけるキャッシュの効果について述べる．

#### 5.4.1　OS 起動時のキャッシュの効果

OS 起動時におけるサーバ OS のキャッシュの効果を確認するために，提案方式でネットワークブートする際，OS が発行したディスクアクセス量と，サーバの実ディスクへ到達したディスクアクセス量を測定した．OS が発行したディスクアクセス量は，vblade が処理したデータ量のログをとることで測定した．サーバの実ディスクへ到達したディスクアクセス量は，サーバ OS である Linux の proc ファイルシステムに記録されるディスクアクセス量を監視して測定した．それぞれをサーバ OS のキャッシュを有効にした場合と，無効にした場合で測定した．測定は，ブートローダである GRUB の選択画面で OS を選択してから，ログイン画面が表示されるまでにかかる区間で行った．

測定結果を**図 6** に示す．各グラフは，OS 起動からのログイン画面表示までの各時間にお


© 2011 Information Processing Society of Japan

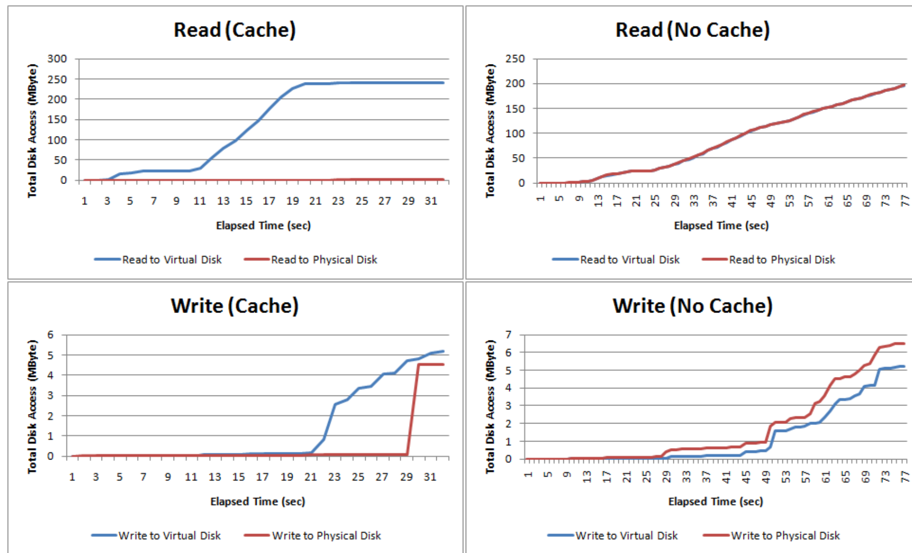


**図 6**　OS 起動中におけるディスクアクセス量解析
Fig. 6　Total amount of transferred data during OS boot up.

ける，ディスクアクセス量の累計を表している．キャッシュを有効にした場合，OS が発行したディスクアクセスの大部分は，サーバの実ディスクに到達していない．これより，ほとんどのディスクアクセスがサーバ OS のキャッシュにヒットし，起動が高速化したと考えられる．なお，キャッシュが有効な場合のライトのグラフにおいて，29 秒付近で，急激にディスクアクセス量が増加しているのは，Linux のキャッシュアルゴリズムがライトバック方式を採用しており，ライトフラッシュが発生しているからと考えられる．

一方，キャッシュを無効にした場合はすべてのディスクアクセスが実ディスクに到達している．これより，キャッシュによる高速化が生じず，ディスクアクセスのネットワーク転送によるオーバヘッドのみが影響して起動時間が延びたと考えられる．また，キャッシュを無効にした場合のライトの測定結果において，OS が発行したディスクアクセス量よりサーバの実ディスクに到達したディスクアクセス量が大きくなっているのは，サーバ OS 上の vblade 以外のアプリケーションによるライトが測定結果に含まれたことによると考えられる．

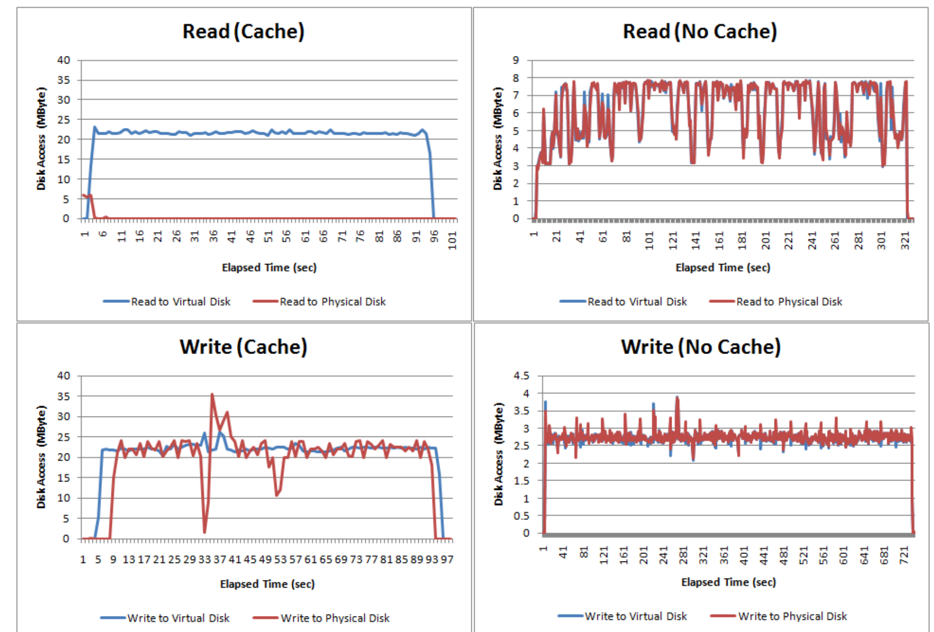


**図 7**　ディスクアクセス速度計測中におけるディスクアクセス量解析
Fig. 7　Amount of transferred data during disk access benchmark.

### 5.4.2　ベンチマーク中のキャッシュの効果

集中的なディスクアクセス時のキャッシュの効果を確認するために，5.3 節でのベンチマーク中において，OS が発行したディスクアクセス量と，サーバの実ディスクへ到達したディスクアクセス量を測定した．5.4.1 項に示した方法と同じ方法を用い，それぞれをサーバ OS のキャッシュを有効にした場合と，無効にした場合で測定した．

測定結果を**図 7** に示す．各グラフは，横軸が経過時間，縦軸が各時間あたりにおけるディスクアクセス量を示す．各グラフは，いずれもシーケンシャルアクセス時における測定結果を示す．なお，図 5 に示されるスループットに比べて，図 7 の各グラフともピークアクセス量が小さく出ているのは，測定のオーバヘッドと考えられる．また，このときのベンチマーク結果は，これらのグラフのピークアクセス量とほぼ同じ値を示している．

リードでは，キャッシュを有効にした場合，OS が発行したディスクアクセスは，ほとんど


© 2011 Information Processing Society of Japan

サーバの実ディスクに届いていないことが分かる．また，ライトでも，OS が発行したディスクアクセス量に対して，サーバの実ディスクに届いたディスクアクセス量が少ない期間が生じている．これより，キャッシュの作用が働いていることが分かる．

ただし，ライトに関しては，リードと比べて，サーバの実ディスクに対して多くのディスクアクセスが生じている．これは，サーバ OS である Linux のキャッシュ設計によるものと考えられる．Linux では，キャッシュされたライトのデータ量が，ある閾値を超えると，キャッシュの内容をいっせいに実ディスクに書き込む．本実験のベンチマーク中では，大量のライトを行っているため，キャッシュされたライトのデータ量がすぐ閾値を超え，頻繁にディスクへの書き込みが生じていると考えられる．

また，Linux ではキャッシュされたライトのデータをいっせいに書き込んでいる間，プロセスの I/O がブロックされる[22]．実際，図 5 においても，キャッシュを有効な状態でネットワークブートした場合では，測定結果のライトのスループットは，リードのスループットより値が低い．キャッシュされたライトのデータ量の閾値を変更することで，ベンチマーク中におけるシーケンシャルライトのスループットが 10 MB/sec 程度変動することが確認できた．

### 5.5　キャッシュ無効時のオーバヘッド

5.3 節において，サーバ OS のキャッシュを無効にした場合，OS をネットワークから起動した場合のディスクアクセスのスループットが大幅に低下する原因を調べるために，ディスクアクセスをサーバへ転送するときの各過程のオーバヘッドを測定した．その結果，サーバアプリケーションである vblade の処理能力がボトルネックになっていることが分かった．ここでは，その根拠となる (1) AoE パケットの送受信にかかるオーバヘッド，(2) サーバ上のディスクアクセスにかかるオーバヘッドの 2 点について述べる．それぞれキャッシュを有効にした場合と，キャッシュを無効にした場合とで比較する．

まず，AoE パケットの送受信にかかるオーバヘッドを調べるために，1 回のディスクアクセスにおける，個々の AoE パケットごとの RTT（Round Trip Time）を測定した．提案方式の実装では，1 回のディスクアクセスでも，転送するデータが大きい場合，複数の AoE 要求パケットに分割されてサーバへ送信される．そこで，1 回のディスクアクセスで生成される要求パケットに対して，クライアント側から送信される順番が早い方から順に 0, 1, 2, ... とシーケンシャル番号を割り当て，それぞれの要求パケットに対する応答パケットが受信されるされるまでの時間を測定した．時間の計測には CPU の TSC（Time Stamp Counter）を利用した．

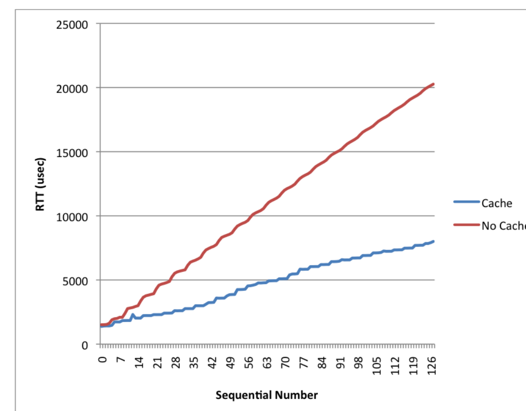


**図 8**　AoE パケットの RTT
Fig. 8　RTT of AoE packets.

5.3 節のディスクアクセスのベンチマークにおいて，レコードサイズ 1024 K のシーケンシャルリードを行っている際の RTT の測定結果を**図 8** に示す．図 8 の RTT の値はディスクアクセス 1000 回のシーケンシャル番号ごとの平均 RTT である．キャッシュの有効・無効にかかわらず，シーケンシャル番号が大きいパケットほど RTT が大きい．つまり，後の方に送信された要求パケットほどサーバ上での処理に時間がかかっていることが分かる．これより，後の方に送信された要求パケットが処理待ちになっている可能性が示唆され，サーバの処理速度がクライアントの要求に追いついていないと考えられる．なお，クライアントマシンとサーバマシンは 1 対 1 で接続されており，両者をつなぐ経路上にボトルネックは存在しないといえる．

図 5 で提案方式のキャッシュを無効にした場合のオーバヘッドは，図 8 における RTT のオーバヘッドで説明できる．すべての要求パケットを送信するのにかかる時間は，RTT に対して非常に小さいため，すべての要求パケットが同時に送信されたと仮定すると，最も大きい RTT の要求パケットが 1 回のディスクアクセスのオーバヘッドといえる．図 8 の差より，キャッシュを無効にした場合は，有効にした場合よりも 1 回のディスクアクセスで 12 ミリ秒のオーバヘッドが加わっている．よって，このオーバヘッドをベンチマーク中に発行されるディスクアクセスの回数分だけ積算した．その結果，キャッシュを無効にした場合は，キャッシュが有効な場合より 49 MB/sec 分スループットが低下するという結果が算


© 2011 Information Processing Society of Japan

出できた．これは，図 5 の測定結果で，提案方式でキャッシュを有効にした場合と無効にした場合のスループットの差にほぼ一致する．また，シーケンシャルライト，ランダムアクセスの場合も同様に一致することが確認できた．

サーバ上でのディスクアクセスにかかるオーバヘッドを調べるために，ディスクアクセスのシステムコール発行時のオーバヘッドを測定した．vblade では，1 つの要求パケットに対してシステムコール（pread() または pwrite()）を呼び出してディスクアクセスを行う．また，vblade はシングルスレッドで動作し，ポーリングによって要求パケットを逐次処理する実装となっている．よって，このときのオーバヘッドを測定するために，1 回のディスクアクセスが転送された際に発行されうる最大回数分システムコールを逐次発行し，すべてのシステムコールが完了するのに要する時間を測定した．その結果，キャッシュが無効な場合は，キャッシュが有効な場合よりも 10〜13 ミリ秒程度も多く処理時間が生じていることが分かった．これは図 8 に見られる，1 回のディスクアクセスの RTT 分のオーバヘッドの差と一致する．

vblade は，シングルスレッドで動作し，ポーリングによって要求パケットを逐次処理する実装となっている．このため，サーバの処理速度がクライアントに追いついていない場合，要求パケットは受信キューで待っていることになり，個々の要求パケットに対するシステムコールの処理時間が積算する．キャッシュが有効である場合は，サーバ上のディスクではなく実質メモリへのアクセスで処理が行われるため，個々のシステムコールの処理時間が非常に小さく，それらが積算されたところで，図 5 で OS をローカルディスクから起動する場合よりスループットが低下することはなかった．一方，キャッシュを無効にした場合は，すべてのシステムコールは直接ディスクへアクセスするオーバヘッドをともない，それらが積算されて非常に大きなオーバヘッドになるため，図 5 で OS をローカルディスクから起動する場合よりスループットが低下したといえる．ただし，レコードサイズが小さいディスクアクセスであれば，1 度に発生する要求パケット数が少ないため，オーバヘッドの積算が抑えられ，スループットの低下が小さくなると考えられる．

キャッシュを無効にした場合におけるディスクアクセスのオーバヘッドは，サーバの処理能力が原因であることから，処理能力の高いサーバを用意することで低減させることができると考えられる．サーバアプリケーションにおいてポーリングではなく，割込みによるパケット受信や，マルチスレッド化によって並列処理を行うことで，キャッシュミスが生じた場合でも，性能低下を抑えることができると考えられる．AoE サーバは，vblade のほかに，より処理能力の高いサーバが存在する．AoE プロトコル自体の実現可能なスループットは非常に大きく，既存の AoE サーバでも，スループットで 1 Gb/sec，10 Gb/sec にも対応している高速なサーバ[23] がある．

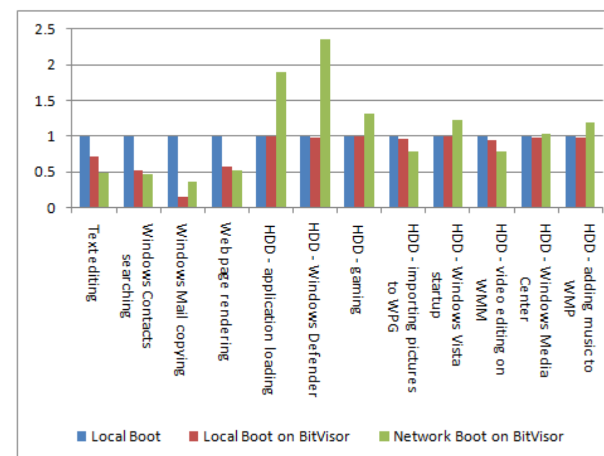


**図 9**　アプリケーションベンチマークの評価結果
Fig. 9　Application benchmark results.

### 5.6　アプリケーションベンチマーク

提案方式で起動した OS 上でアプリケーションベンチマークを実施した．実験には PCMark Vantage[24] を使用した．PCMark Vantage は，ユーザが日常に PC を利用するケースを想定し，複数のアプリケーションを動作させ，そのときのシステムの処理能力を評価する．比較のため，提案方式に加え，OS をローカルディスクから起動した場合，BitVisor 上で OS をローカルディスクから起動した場合も測定した．

測定結果を**図 9** に示す．図 9 は，各項目において，ローカルディスクから起動した場合の評価値を 1 としたとき，他の場合の評価値を比で表したものである．値が大きいほど，性能が高いことを示している．アプリケーションの起動，ウィルススキャン，動画や音声の編集再生，画像や音楽の読み込みといった項目でローカルディスクから起動した場合と同等またはそれ以上の評価値となった．特に，Windows Defender Scan や Application Loading，Add Musics to WMP などディスクアクセス処理の評価が多く含まれるケースがキャッシュの影響で評価値が高い傾向にある．よって，これらのケースでは，提案方式はローカルディスクから起動した場合に比べて，性能面で透過的であるといえる．一方，Web Page Rendering,


© 2011 Information Processing Society of Japan

Windows Mail Searching，Windows Mail Copying ではローカルディスクから起動した場合に比べて提案方式の評価値が低い．これらのケースでは大量のデータをメモリ上に展開するという処理が評価されるという点で共通している[25]．これより，評価値が低下した理由は，BitVisor のメモリ管理におけるオーバヘッドが大きいからと考えられる．BitVisor は大量にメモリをアロケートするような処理や，ページフォルトが頻発する処理でオーバヘッドが大きくなる[5]．これより，提案方式は，大量のメモリアクセスを行わない場合のみ，ローカルディスクから起動した場合と比べて性能面でも透過的であるといえる．ただし，BitVisor の実装で EPT（Extended Page Table）などのハードウェアによるメモリ管理支援機能を利用することで性能改善が図れるため，メモリ管理のオーバヘッドは将来的に改善される可能性がある[26]．

### 5.7　既存の仮想マシンモニタ方式との比較

提案方式が，仮想マシンモニタを用いた既存のネットワークブート方式（2.2 節参照）よりもオーバヘッドが小さいことを確かめるために，VMM および OS の起動時間，ディスクアクセスのスループットについて，既存の仮想マシンモニタ方式と性能を比較した．既存の仮想マシンモニタ方式として KVM と NFS を用いて OS をネットワークブートした場合を測定した．KVM には，ホスト OS として Debian 6.0（Squeeze），ディスクアクセスの転送に NFS 3.0 を用いた．ただし，NFS は，デフォルトの設定ではホスト OS のメモリがキャッシュとして利用される．よって，今回の実験では，対等な比較を行うために，NFS によるホスト OS 側のメモリキャッシュを無効にして測定した．また，NFS サーバのパケット送受信キューのサイズを AoE サーバと同等に設定した．以上の設定は，ホスト OS 上で NFS をマウントする際にオプションとして-o noac，rsize=4096，wsize=4096 と指定することで行った．また，動作する NFS サーバスレッドの数も AoE サーバと同等にするために，nfsd 1 というコマンドをサーバ OS 上で実行した．なお，いずれの方式もサーバ OS のキャッシュは有効にして測定を行った．

VMM および OS の起動時間を測定した．BitVisor，KVM の VMM 自体の起動時間および，その上でネットワークブートさせた OS の起動時間を測定した．VMM 自体の起動時間は PC の電源入力時から，OS 起動直前までにかかる時間を測定した．また，OS の起動時間は，VMM によって OS の起動が開始されてからログイン画面までにかかる時間をストップウォッチを用いて測定した．測定結果を**図 10** に示す．BitVisor の VMM 自体の起動時間は，KVM よりも 20 秒短い．また，OS の起動時間も，BitVisor 上でネットワークブートした場合のほうが，KVM 上よりも 12 秒短い．これより，提案方式は，KVM による方式に比べてより透過的にネットワークブートできるといえる．

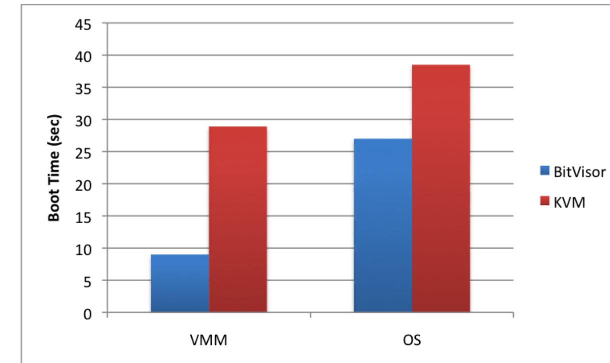


**図 10**　既存仮想マシンモニタ方式との起動時間の比較結果
Fig. 10　VMM boot time comparison with KVM.

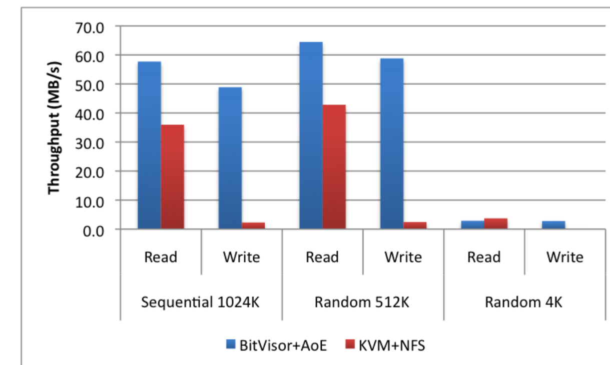


**図 11**　既存仮想マシンモニタ方式とのディスクアクセス速度の比較結果
Fig. 11　Disk access throughput comparison with KVM.

また，BitVisor，KVM 上でネットワークブートした OS 上で，5.3 節と同様にディスクアクセスのスループットを測定するベンチマークを行った．測定結果を**図 11** に示す．提案方式でのディスクアクセスのスループットは，シーケンシャルリードでは，KVM 上に比べて 21.8 MB/sec スループットが大きい．また，ライトでは，KVM に比べて 46.6 MB/sec



© 2011 Information Processing Society of Japan

スループットが大きい．一方，レコードサイズ 512 K のランダムリードでは，KVM に比べて 21.6 MB/sec スループットが大きい．また，ライトでは，KVM に比べて 56.1 MB/sec スループットが大きい．これより，提案方式は，KVM を用いた既存の仮想マシンモニタ方式と比較して同等以上のスループットを得ており，より少ないオーバヘッドで動作できるといえる．

## 6. まとめと今後の課題

本論文では，仮想マシンモニタを用いた透過的なネットワークブート方式を提案した．提案方式では，クライアント端末に仮想マシンモニタを導入し，ゲスト OS が通常の PC におけるローカルディスクと同様の方式でストレージにアクセスできるようにしつつ，そのアクセスをネットワーク経由でサーバに転送した．これにより，既存の PC 向けの OS をいっさい改変することなくネットワークブートすることが可能になった．

提案方式を準パススルー型仮想マシンモニタ BitVisor をベースに実装し，実際に既存の OS である，Linux（Debian）や Windows（Vista，7）をネットワークブートできることを確認した．また，サーバ上のキャッシュの影響により，OS をローカルディスクから直接起動した場合と比べて，起動時間が 14 秒短縮された．キャッシュの影響を除いても，24 秒程度の増加に抑えられた．また，アプリケーションベンチマークの結果，メモリ管理が大量に発生するユースケースを除き，OS をローカルディスクから直接起動した場合と同等の評価結果を得た．

今後の課題としては，クライアント端末台数が増加によるサーバへの負荷集中を抑えるため，クライアント端末側でローカルディスクをキャッシュとして利用することが考えられる．

**謝辞**　本研究は科研費（22700022）の助成を受けたものである．

## 参考文献

1) Chandra, R., Zeldovich, N., Sapuntzakis, C. and Lam, M.S.: The Collective: A Cache-Based System Management Architecture, *Proc. 2nd Conference on Symposium on Networked Systems Design and Implementation*, pp.259–272 (2005).
2) Eisler, M., Labiaga, R. and Stern, H.: *Managing NFS and NIS, 2nd Edition*, O'Reilly Media (2001).
3) Ardence, available from ⟨http://www.ardence.com/⟩.
4) Kozuch, M. and Satyanarayanan, M.: Internet Suspend/Resume, *Proc. 4th IEEE Workshop on Mobile Computing Systems and Applications*, p.40 (2002).
5) Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E., Kono, K., Chiba, S., Shinjo, Y. and Kato, K.: BitVisor: A Thin Hypervisor for Enforcing I/O Device Security, *Proc. 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (*VEE 2009*), pp.121–130 (2009).
6) Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D. and Lyon, B.: Design and Implementation of the Sun Network Filesystem, *Proc. USENIX Summer Conference* (1985).
7) Howard, J.H., Kazar, M.L. and Menees, S.G.: Scale and Performance in a Distributed File System, *ACM Trans. Computer Systems*, Vol.6, No.1, pp.51–81 (1988).
8) Satran, J., Meth, K., Sapuntzakis, C., Chadalapaka, M. and Zeidner, E.: RFC 3720: Internet Small Computer Systems Interface (iSCSI) (2004).
9) Hopkins, S. and Coile, B.: AoE (ATA over Ethernet), Specification (2009).
10) Breuer, P.T., Lopez, A.M. and Ares, A.G.: The network block device, *Linux Journal* (2000).
11) Zhou, Y., Zhang, Y. and Xie, Y.: Virtual Disk Based Centralized Management for Enterprise Networks, *Proc. 2006 SIGCOMM Workshop on Internet Network Management* (2006).
12) DistroWatch.com: Put the fun back into computing. Use Linux, BSD.
13) SOL/IDE-R, available from ⟨http://software.intel.com/en-us/blogs/2007/01/08/sol-ider-explained/⟩.
14) 瀬戸洋一，織田昌之，寺田真敏：情報セキュリティ概論，日本工業出版 (1997).
15) Baratto, R.A., Kim, L.N. and Nieh, J.: THINC: A Virtual Display Architecture for Thin-Client Computing, *Proc. 20th ACM Symposium on Operating Systems Principles* (2005).
16) Parichha, B.K. and A., G.T.: Remote Device Support in Thin Client Network, *Proc. 3rd Annual ACM Bangalore Conference*, pp.1–4 (2010).
17) Lai, A.M. and Nieh, J.: On the Performance of Wide-Area Thin-Client Computing, *ACM Trans. Computer Systems* (*TOCS*), Vol.24, No.2, pp.157–209 (2006).
18) ATA over Ethernet Tools, available from ⟨http://aoetools.sourceforge.net/⟩.
19) gPXE, available from ⟨http://etherboot.org/⟩.
20) Ben-Yehuda, M., Day, M.D., Dubitzky, Z., Factor, M., Har'El, N., Gordon, A., Liguori, A., Wasserman, O. and Yassour, B.-A.: The turtles project: Design and implementation of nested virtualization, *Proc. 9th USENIX conference on Operating systems design and implementation*, OSDI'10, Berkeley, CA, USA, USENIX Association, pp.1–6 (2010).
21) Crystal Dew World, available from ⟨http://crystalmark.info/⟩.
22) Bovet, D.P. and Cesati, M.: *Understanding the Linux Kernel, 3rd Edition*,



© 2011 Information Processing Society of Japan

O'REILLY (2005).
23) Coraid, available from ⟨http://www.coraid.com/⟩.
24) Futuremark, available from ⟨http://www.futuremark.com/⟩.
25) Futuremark Corporation: PCMark Vantage Whitepaper v1.0 (2007).
26) Performance Evaluation of Intel EPT Hardware Assist, available from ⟨http://www.vmware.com/pdf/Perf_ESX_Intel-EPT-eval.pdf⟩.

(平成 23 年 1 月 28 日受付)
(平成 23 年 6 月 8 日採録)

**表　　祐志**（正会員）

1987 年生まれ．筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻博士前期課程所属．オペレーティングシステムや仮想マシンモニタ等のシステムソフトウェアに興味を持つ．

**品川　高廣**（正会員）

1974 年生まれ．2003 年東京大学大学院理学系研究科情報科学専攻博士課程修了，博士（理学）取得．同年東京農工大学工学部情報コミュニケーション工学科助手，2007 年筑波大学大学院システム情報工学研究科講師，2011 年より東京大学情報基盤センター情報メディア教育研究部門准教授．オペレーティングシステムや仮想マシンモニタ等のシステムソフトウェアに興味を持つ．平成 11 年度情報処理学会論文賞，平成 14 年度山下記念研究賞受賞．

**加藤　和彦**（正会員）

1962 年生まれ．1985 年筑波大学第三学群情報学類卒業．1989 年東京大学大学院理学系研究科博士課程中退．1992 年博士（理学）（東京大学）．1989 年東京大学理学部情報科学科助手，1993 年筑波大学電子・情報工学系講師，1996 年同助教授，2004 年より筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻教授．オペレーティングシステム，分散システム，仮想計算環境，セキュリティに興味を持つ．1990 年情報処理学会学術奨励賞，1992 年同研究賞，2005 年同論文賞，2004 年日本ソフトウェア科学会論文賞各受賞．



© 2011 Information Processing Society of Japan