iSCSIターゲットソフトウェアの解析

藤田智成[†] 小河原成哲[†]

Ethernet 機器を用いて，SAN（Storage Area Network）を実現する iSCSI プロトコルが注目を浴びている．これまでに，iSCSI プロトコルの性能に関する複数の報告がされているが，それらは SCSI コマンドを発行する計算機（イニシエータ）を対象としており，SCSI コマンドを処理するストレージシステム（ターゲット）の設計の妥当性や性能に関する議論はされていない．本稿では，iSCSI プロトコル用ストレージシステムを実現するために使われる 2 種類のオープンソースソフトウェアを用いて，(1) 汎用オペレーティングシステムのカーネルが提供する標準的機能を利用する手法，(2) 変更を加えたカーネルが提供する iSCSI ストレージシステムに最適化された機能を利用する手法，それぞれが性能に与える影響を解析した．性能評価では，後者が前者よりも優れていることを確認した．しかし，設計に変更を加えることで，前者の標準的なインタフェースだけを用いる手法でも，一般の作業負荷を模擬したベンチマークで，後者と同等の性能を達成可能であることを確認した．

Analysis of iSCSI Target Software

Tomonori Fujita[†] and Masanori Ogawara[†]

Many companies have started to consider the iSCSI protocol used to build inexpensive SAN (Storage Area Network) environment, which provides management advantages, over Ethernet networks. Several studies have conducted the performance evaluation of the iSCSI protocol. However, all of them evaluated the performance of a host computer, called an initiator, issuing SCSI commands. The performance of storage systems providing services to initiators have not been investigated. This paper analyzes the design and performance of open-source software for building iSCSI storage systems with a general purpose operating system. Our experiments shows, compared with the storage system using only general functions provided by the operating system, the storage system using specialized functions provided by the modified operating system gives better performance. However, our results also show that careful design enables the storage system using only general functions to provide comparable performance that the storage system using specialized functions provides.

1. はじめに

急速なデータ容量の増大に対応するため，多くの企業は，計算機とストレージシステム間を専用の高速なネットワークで接続する，Storage Area Network（SAN）と呼ばれるストレージアーキテクチャの導入を進めている．計算機とストレージシステムがシステムバスで直接に接続される従来のストレージアーキテクチャ，Direct Attached Storage（DAS）と比較して，SAN はストレージ容量の拡大，遠隔地へのバックアップ等，その管理が容易であるという利点を持つ．

従来，Fibre Channel（FC）と呼ばれる高速ネットワーク技術が，SAN の構築のために用いられてきた．FC と広く普及しているネットワーク技術である Ethernet を比較すると，FC 用の NIC やスイッチは高価であり，その点が SAN 導入の障壁となっていた．この問題に対して，高価な SAN 機器の代わりに，安価な Ethernet 機器を利用して実現する IP-SAN が提案され，注目を浴びている．

IP-SAN で用いられる代表的なストレージプロトコルである iSCSI プロトコル[1] は，計算機とストレージシステムを Ethernet で接続し，SCSI コマンドを TCP/IP パケットに包んで転送する．iSCSI プロトコルは，計算機とストレージを接続している SCSI ケーブルを Ethernet ケーブルに置き換える．計算機は遠隔のディスクを，システムバス経由で接続されたディスクと同様に認識する．

これまでに，iSCSI プロトコルに関して，複数の性能評価結果が報告されている．しかし，それらは，SCSI コマンドを発行し，ストレージシステムにサービスを要求する計算機，つまりイニシエータを対象としてい

† NTT サイバーソリューション研究所
NTT Cyber Solutions Laboratories

る．一方，イニシエータから SCSI コマンドを受け取り，計算機にサービスを提供するストレージシステム，つまり，ターゲットの設計手法の妥当性に関する議論や性能評価はされていない．

本稿では，PC-AT 互換機用の汎用のオペレーティングシステム上で動作し，iSCSI ターゲット機能を実現する 2 種類のオープンソースソフトウェアの設計手法を評価した．

商用の iSCSI ターゲットシステムは，iSCSI プロトコルを高速に処理するための専用 NIC や特殊なオペレーティングシステムを用いる．一方，本稿では，汎用の PC と Ethernet NIC，オープンソースのオペレーティングシステムを用いて実装した iSCSI ターゲットシステムに解析の焦点をあてる．

性能面では，商用システムが用いる手法が有利である．しかし，後者が利用する，PC 向けの汎用ハードウェアの性能は急速に進歩しており，低価格化も著しい．加えて，オープンソースのオペレーティングシステムは，その機能面で，急速な進歩を見せている．したがって，後者の手法は，コスト，機能改善の速度の点で有利である．

解析した 2 種類のオープンソースの iSCSI ターゲットソフトウェアは，(1) カーネルが提供する標準的な機能を利用する，(2) 変更を加えたカーネルと独自に実装した機能を利用する，という異なる方針に基づいて，実装されている．

我々の実験では，後者の設計方針に基づいて設計されたターゲットの性能は，前者の性能を大きく上回ることが明らかになった．しかし，設計を変更することで，前者の設計手法でも，一般作業負荷に対し，後者と同等の性能を提供するターゲットを実現することができた．また，その性能は，エントリクラスの商用 iSCSI ターゲットシステムの性能に匹敵することを確認した．

我々は，2 種類の Linux 用 iSCSI ターゲットソフトウェアを用いた．1 つはニューハンプシャ大学とヒューレット・パッカード社が共同で実装した UNH ターゲット[2]．もう一方は，Ardis Technologies 社が実装した Ardis ターゲット[3] である．本稿では，UNH ターゲットのバージョン 1.5.03 と，Ardis ターゲットのバージョン 20040211 を用いた．

本稿の構成は以下のとおりである．まず，2 章で関連研究をまとめる．次に，3 章で，2 種類の iSCSI ターゲットソフトウェアと我々が新たに実装したターゲットソフトウェア，各々の設計を比較する．4 章では，ターゲットの性能を比較し，設計の違いが性能に与える影響を解析する．最後に 5 章でまとめる．

2. 関連研究

過去に，iSCSI プロトコル用ターゲットの性能報告はされていないが，iSCSI プロトコルの性能特性，イニシエータの実装手法やネットワーク条件が性能に与える影響に関して，複数の報告がされている．

文献 4) において，Radkov らは，NFS プロトコルを用いた NAS（Network Attached Storage）と，iSCSI プロトコルを用いた IP-SAN の性能比較を行っている．また，文献 5) において，Aiken らは，FC を用いた SAN と，iSCSI プロトコルを用いた IP-SAN の性能を比較している．

文献 6) において，Sarkar らは，iSCSI プロトコル専用 NIC と汎用の Ethernet NIC を用いたイニシエータの性能差を比較している．

文献 7) で，Ng らは，ネットワーク帯域と遅延が iSCSI プロトコルの性能に与える影響を報告している．

文献 8) で，Palekar らは，UNH ターゲットの設計について説明しているが，その設計と性能の関係については考察していない．

我々が用いた 2 種類のソフトウェア以外に，Intel 社がオープンソースのターゲットソフトウェア[9] を公開している．しかし，Intel 社のターゲットはユーザ空間で動作するため，カーネル空間で動作する UNH や Ardis ターゲットに比べて，ユーザ・カーネル間で発生するメモリコピー，システムコール等のオーバヘッドが大きく，その実用性が低いため，本稿では解析の対象から外した．

3. ターゲットソフトウェアの設計

UNH ターゲットは，ファイルモード，ディスクモードと呼ばれる，まったく異なる設計がされた，2 種類の動作方式を実装している．本稿では，前者の方式で動作させたターゲットを UNH ファイルモードターゲット，後者を UNH ディスクモードターゲットと呼ぶ．両方のターゲットを指すときは，単に UNH ターゲットと記すことにする．

UNH ファイルモードターゲット，UNH ディスクモードターゲット，Ardis ターゲット，加えて，我々が新たに設計したターゲットソフトウェア（Threaded-Ardis ターゲットと呼ぶ），合計 4 種類のターゲットソフトウェアの設計を比較する．

UNH ターゲットは，カーネルが提供する標準的な機能を利用する方針に基づいて設計されている．一方，Ardis ターゲットは，変更を加えたカーネルと独自に

実装した機能を利用する方針に基づいて設計されている．

Threaded-Ardis ターゲットは，UNH ターゲットと同様に，カーネルが提供する標準的な機能を利用する方針に基づいて設計されている．我々は，Ardis ターゲットの独自に実装した機能と変更されたカーネルが提供する機能を利用して実装されている部分を，カーネルの標準的な機能を用いた実装で置き換え，Threaded-Ardis ターゲットを実装した．

我々が，新たにターゲットソフトウェアを実装するにあたり，UNH ターゲットではなく，Ardis ターゲットのコードを用いた理由は，(1) Ardis ターゲットの設計が簡素で（後述するようにディスク入出力部分を除く），コードの再利用が容易である．(2) 複数のイニシエータに，個別のディスクイメージを提供できる等の商用環境での運用に必要な機能を想定した設計がされている，という 2 点である．

3.1 評 価 基 準

iSCSI ターゲットソフトウェアの設計手法の評価基準を以下の 4 項目とした．

カーネルへの変更の有無 Linux カーネルのソースコードへの変更が必要であるかを評価基準の 1 つとした．iSCSI ストレージシステムとしての運用に最適化されていない Linux カーネルを改良することで，性能向上が期待される．しかし，カーネルへの独自の変更は，汎用性を失わせ，カーネル開発者が今後行う改善，新機能を利用することが困難になるため，望ましくない．

DAS 環境との互換性 iSCSI ターゲットがイニシエータに提供するディスクが，計算機に直接接続される従来のディスクとまったく同じように動作するかを解析した．互換性を持たないターゲットを利用する場合，その動作に合わせて既存のアプリケーションへの変更が必要となるため，その導入コストは大きく上昇する．

ディスク管理機能 Linux カーネルが提供するディスク管理機能との連動が可能な設計であるかを評価した．Logical Volume Manager [10] が提供する，ディスク容量の動的な増減等の柔軟な管理機能，Software RAID が提供する複数ディスクを用いた冗長性機能等を持つ，仮想的なディスクドライブを，イニシエータに提供するディスクとして利用できることが望ましい．

性能 商用環境で発生する負荷を処理可能な性能を実現できる設計であるかを評価した．本稿では，実環境での負荷を模擬するベンチマークを用いて，商用の iSCSI ターゲットシステムとの性能比較を行った．

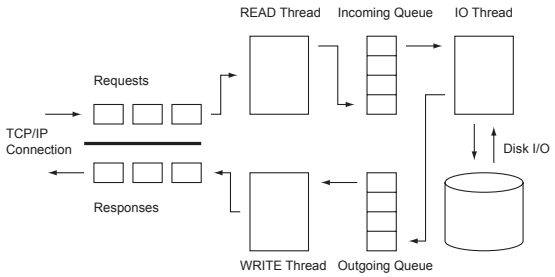

図 1 iSCSI ターゲットソフトウェアの構成
Fig. 1 Architecture overview of iSCSI target software.

3.2 全 体 構 成

すべてのソフトウェアともに，3 種類のカーネルスレッドが，iSCSI ターゲットの主要な機能を提供する．本稿では，3 種類のスレッドを，**図 1** に示すように，READ スレッド，WRITE スレッド，IO スレッドと呼ぶ．

READ スレッドは，イニシエータから送られてくる iSCSI Protocol Data Unit（PDU）をソケットから読み出す．PDU は，イニシエータとターゲットの間でやりとりされるメッセージの最小単位で，iSCSI ヘッダと，SCSI コマンドから構成されている．PDU がディスク IO 命令の実行を必要とする SCSI コマンドを含んでいる場合は，READ スレッドはそのコマンドを Incoming キューにつなぐ．

IO スレッドは，ディスクに対するデータ入出力を担当する．IO スレッドは，Incoming キューから，SCSI コマンドを取り出し，適切なディスク IO 命令を実行する．SCSI コマンドが要求したディスク IO 命令が完了すると，IO スレッドは，SCSI コマンドを Outgoing キューにつなぐ．

WRITE スレッドは，Outgoing キューにつながれている SCSI コマンドを取り出し，その結果を新しい PDU に入れ，イニシエータに送信する．

3.3 IO スレッドの設計

4 種類のターゲットの設計が最も異なる部分が，IO スレッドである．Linux カーネル内には，ディスク入出力命令の実行に用いることのできるインタフェースが複数あり，4 つのターゲットの IO スレッドは，異なるインタフェースを用いている．その設計の違いを**図 2** に示す．

UNH ファイルモードターゲットと Threaded-Ardis ターゲットは，Virtual File System（VFS）サブシス

表 1 iSCSI ターゲットソフトウェアの特徴
Table 1 iSCSI target software characteristics.

	標準カーネル利用	DAS 環境互換性	ディスク管理機能	性能
UNH ファイルモード	○	×	○	×（コマンド並列処理不可）
UNH ディスクモード	○	○	×	×（ページキャッシュ利用不可）
Ardis	×	○	○	○
Threaded-Ardis	○	○	○	○

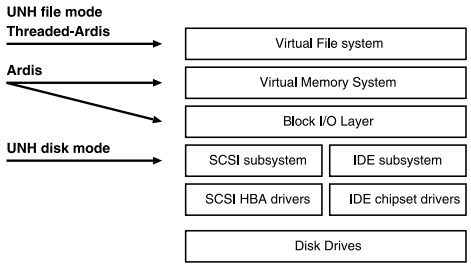

図 2 IO スレッドが用いるカーネル IO インタフェース
Fig. 2 IO interfaces.

テムが提供するカーネルの標準インタフェース☆を用いて，ディスク IO 命令を実行する．

カーネル標準ファイルインタフェースは，通常のファイルとブロックデバイス（デバイスファイルを経由して）へのアクセスを提供する．しかし，UNH ファイルモードターゲットは通常ファイルのみ，Threaded-Ardis ターゲットはブロックデバイスだけを，イニシエータにディスクイメージとして提供することが可能である．

UNH ディスクモードターゲットは，SCSI ホストバスアダプタを管理する SCSI サブシステムが提供するカーネル標準インタフェース☆☆を利用する．PDU に含まれている SCSI コマンドが，カーネル標準 SCSI インタフェースに渡され，ディスク IO 命令が実行される．したがって，SCSI サブシステムが管理するディスクのみ，つまり，ホストに接続された SCSI ディスクだけをイニシエータにディスクイメージとして提供することができる．

Ardis ターゲットは，ブロックデバイスを管理するブロック I/O レイヤと変更を加えた仮想記憶（Virtual Memory）システムが提供する機能を組み合わせ，ディスク入出力を実行する．Ardis ターゲットは，ブロックデバイスをイニシエータにディスクイメージとして提供することができる．

☆ 主に，ファイルの読み書き用のシステムコールが，このインタフェースを利用する．以降，このインタフェースをカーネル標準ファイルインタフェースと呼ぶ．

☆☆ 以降，このインタフェースをカーネル標準 SCSI インタフェースと呼ぶ．

3.1 節の評価基準を用いて，IO スレッドの設計の違いによって生じる，それぞれのターゲットの特徴を**表 1** にまとめた．

3.3.1 カーネルへの変更の有無

Ardis ターゲットだけが，Linux カーネルのソースコードへの変更を必要とする．

UNH ファイルモードターゲット，UNH ディスクモードターゲットと Threaded-Ardis ターゲットは，カーネルの各サブシステムが公開しているデータ構造と関数，つまり，カーネルの標準のインタフェースだけを用いて，他のサブシステムと連動し，必要な機能を実現している．

一方，Ardis ターゲットは，カーネルのソースコードを変更することで，カーネルの各サブシステムが外部に公開していないデータと関数を用いている．したがって，Ardis ターゲットを動作させるためには，変更したカーネルソースコードから再構築されたカーネルイメージが必要である．

前述したように，カーネルへの変更は互換性を失わせ，他の開発者による今後の Linux カーネルのバグ修正，改良，新機能を利用することを困難にする．また，独自の変更を加えたことで，カーネルの安定性が損なわれる可能性もある．カーネルへ変更を加えることは，開発・保守コストの上昇につながるといえる．

開発・保守コストに関して，Ardis ターゲットとその他のターゲットの設計に関して，考慮すべき違いがもう 1 点ある．

UNH ファイルモードターゲット，UNH ディスクモードターゲットと Threaded-Ardis ターゲットは，高レベルなカーネル標準インタフェース☆☆☆を多用することで，その設計，実装を簡略化している．

一方，Ardis ターゲットは，高レベルなカーネル標準インタフェースを使わずに，単一の基本的な操作だけを行う，低レベルな標準インタフェースを組み合わせることで，必要な機能を実現している．

UNH ファイルモードターゲットと Threaded-Ardis

☆☆☆ 本稿では，高レベルな関数とは，多くの操作を必要とする複雑な機能を実現する関数を意味する．

ターゲットが利用するカーネル標準ファイルインタフェースは，データの読み書きに使うメモリのアドレス，データ長，オフセット（ディスク位置）を指定するだけで，ディスク IO 命令を実行することができる．

同様に，UNH ディスクモードターゲットが利用する，カーネル標準 SCSI インタフェースも，データのメモリアドレス，データ長，ディスク位置を指定し，ディスク IO 命令を実行することができる．

Ardis ターゲットは，単一の操作だけを実行する低レベルな標準インタフェースを利用するため，ディスク入出力は複雑化する．Ardis ターゲットが独自に実装したディスク入出力関数は，以下のような操作を実行する．

- データの読み書きに使うページキャッシュの確保（仮想記憶）
- ページキャッシュの参照カウンタを増やす（仮想記憶）
- ページキャッシュのロック（仮想記憶）
- ブロックディスクへの IO 命令要求（ブロック I/O レイヤ）
- IO 命令完了にともなって呼ばれる割込みハンドラによるページキャッシュのロック解放（ブロック I/O レイヤ）
- ページキャッシュの参照カウンタを減らす（仮想記憶）

低レベルな標準インタフェースを多用することは，カーネルへ変更を加えること同様に，保守・開発コストを上昇させる．

カーネルへの変更や低レベルな標準インタフェースの多用に関しては，それらの欠点である保守・開発コスト増加の程度と，得られる性能上の効果の両方を考慮することが重要である．

3.3.2 DAS 環境との互換性

UNH ファイルモードターゲットは，書き込み処理に関して，従来の DAS 環境のディスクとは互換性のない動作を行う．

ディスクに直接アクセスする既存のアプリケーション（ファイルシステムやデータベース等）は，書き込み命令を発行し，ディスクからその完了通知を受け取った時点で，データのディスクへの書き込みが終了していることを想定している．たとえば，多くのファイルシステムが用いている，停電等の障害発生時のデータ破損を回避するためのジャーナリング技術[11] は，上記のディスクの動作に依存している．

しかし，UNH ファイルモードターゲットは，イニシエータから受け取った書き込みデータをページキャッシュにコピーし終えた時点で，完了通知を送信し，更新したページキャッシュとディスクの同期に関して何もしない．通常，それらのページキャッシュは，更新されてから 30 秒以上経過した後に，kupdate カーネルスレッド☆によって，ディスクと同期される．その前に，停電等の予期せぬ障害が発生すると，イニシエータが完了通知を受け取った書き込み命令のデータが，実際にはディスクに保存されていない状況が発生する．

加えて，イニシエータにとって，書き込んだデータがターゲットのディスクに保存されたかどうかを知る方法が存在しないため，この動作のもとで，障害時にデータ破損を回避することは不可能である．したがって，障害時の対策が不可欠である商用環境で，UNH ファイルモードターゲットを運用することは現実的ではない．

UNH ファイルモードターゲットが，このように動作するのは，性能上の理由からだと考えられる．カーネル標準ファイルインタフェースは，ダーティなページキャッシュとディスクを同期する sys_fsync 関数☆☆ を提供する．しかし，sys_fsync 関数は，指定したデバイスのダーティなページキャッシュすべてをディスクと同期するため，ターゲットがこの関数を呼び出すと，複数の書き込みコマンドを処理している場合に，すぐに同期する必要がないページキャッシュまで同期するため，効率が悪い．

UNH ファイルモードターゲットと同様に，Threaded-Ardis ターゲットも，カーネル標準ファイルインタフェースを用いて，書き込みを処理する．しかし，ダーティなページキャッシュとディスクの同期操作に関しては，低レベルなカーネル標準関数を利用して，実現している．この関数は，指定した範囲のページキャッシュだけをディスクを同期することができる．

4 章の性能評価で，UNH ファイルモードターゲットを，sys_fsync 関数を使うように変更し，その性能への影響を確認する．

3.3.3 ディスク管理機能

UNH ファイルモードターゲット，Ardis ターゲット，Threaded-Ardis ターゲットは Linux カーネルが提供するデバイスの仮想化機能を利用することができる．一方，UNH ディスクモードターゲットは，SCSI ディスクしか利用することができないという欠点を持つ．

Linux カーネルでは，ブロック I/O レイヤが，SCSI ディスクや IDE ディスク等の様々な物理インタフェー

☆ システムのダーティなページキャッシュとディスクを定期的に同期する作業を担当するプロセス．

☆☆ この関数は，fsync システムコールの実装に使われている．

スのディスクにアクセスするための，統一されたインタフェースを提供する．また，マルチデバイスサブシステムが，柔軟な管理機能や冗長性機能を持った仮想的なブロックデバイスを実現する．

UNH ディスクモードターゲットが利用する SCSI サブシステムは，上記 2 つのサブシステムよりも低レイヤに位置する（物理デバイスに近い）．したがって，カーネル標準 SCSI インタフェースによって起動されたディスク IO 命令は，それらのサブシステムを通過せずに，SCSI ディスクに到達する．

3.3.4 性　　能

UNH ディスクモードターゲットが持つ性能上の欠点は，仮想記憶システムが提供するページキャッシュ機能を利用できないことである．

ページキャッシュは，一度アクセスされたデータを，次のアクセス時に備えてメインメモリに保持するキャッシュ機能を提供する．UNH ディスクモードターゲットの利用する SCSI サブシステムは，仮想記憶システムよりも低レイヤに位置する．したがって，カーネル標準 SCSI インタフェースによって起動されたディスク IO 命令は，仮想記憶システムのページキャッシュを経由せずに SCSI ディスクに到達する．その結果，イニシエータからの読み込み命令のすべてに対して，実際にディスク IO 命令が実行される．このことは，同じディスクブロックに対して何度も読み込みが発生する負荷に対する性能を低下させる．4 章の性能評価で，この点を確認する．

UNH ファイルモードターゲットの設計には，イニシエータから送られてきた SCSI コマンドを並列処理することができないという性能上の欠点がある．

高性能なターゲットを実現するためには，イニシエータから次々と送られてくる SCSI コマンドを並列に（同時に）処理する必要がある．そのためには，IO スレッドがスリープ（ブロック）することなく，複数のディスク IO 命令を実行しなければならない．IO スレッドがスリープしている間は，後に続く SCSI コマンドの処理が停止するためである．

UNH ファイルモードターゲットと Threaded-Ardis ターゲットが用いる，カーネル標準ファイルインタフェースは同期 IO 操作であるため，並列処理の実現は困難である．つまり，このインタフェースを呼び出したプロセスは，IO 命令が完了するまで，スリープする．

Threaded-Ardis ターゲットは，複数の IO スレッドを利用することで，上記の問題を解決している．IO スレッドは，ディスク IO 命令が完了するまでスリープするが，複数の IO スレッドが存在するため，ターゲットは同時に複数の SCSI コマンドを処理できる．

UNH ディスクモードターゲットの場合，利用しているカーネル標準 SCSI インタフェースが非同期 IO 操作であるため，容易に SCSI コマンドの並列処理を実現できる．IO スレッドは，ディスク IO 命令完了にともない非同期に呼び出される関数を指定し，スリープすることなく，ディスク IO 命令を開始することができる．

Ardis ターゲットの IO スレッドの場合，ディスク IO を起動するために用いるブロック I/O レイヤが提供するカーネル標準インタフェースは非同期 IO 操作である．しかし，仮想記憶システムが提供するページキャッシュ操作のインタフェースが非同期に動作しないため，SCSI コマンドの並列処理を容易に実現できない．

Ardis ターゲットは，SCSI コマンドの並列処理を実現するため，仮想記憶システムを変更し，ページキャッシュをロックできない場合☆に，呼び出したプロセスをスリープさせずにエラーを返す関数を追加している．加えて，非公開の関数を利用し，指定した複数のページキャッシュのうち，いずれかのロックが取得できるまでプロセスがスリープする機能を実現している．

3.4 書き込み処理

iSCSI プロトコルは，データの流れをイニシエータ中心に表記し，書き込みはイニシエータからターゲット方向へのデータの移動，つまり，ターゲットのディスクへのデータの書き込みを意味する．

iSCSI プロトコルの書き込みは，Solicited Data，Unsolicited Data の 2 種類に分類することができる．

Solicited Data 方式では，イニシエータは WRITE コマンドを送信し，ターゲットから Ready to transfer（R2T）コマンドを受信した後に，書き込みデータの送信を開始する．

Unsolicited Data 方式では，R2T コマンドは使われず，イニシエータは WRITE コマンドに続けて，書き込みデータを送信することができる．Unsolicited Data 方式は，WRITE コマンド，データをそれぞれ個別の PDU で送信する通常方式と，WRITE コマンドと書き込みデータを 1 つの PDU にまとめて送信する方式（Immediate Data）を定義している．

Solicited Data 方式，Unsolicited Data 方式の通常型と Immediate Data 型，合計 3 種類の書き込み方

☆ ディスクからページキャッシュへのデータ読み込み操作（READ コマンド），ページキャッシュとディスクの同期操作（WRITE コマンド）は，ページキャッシュをロックする必要がある．

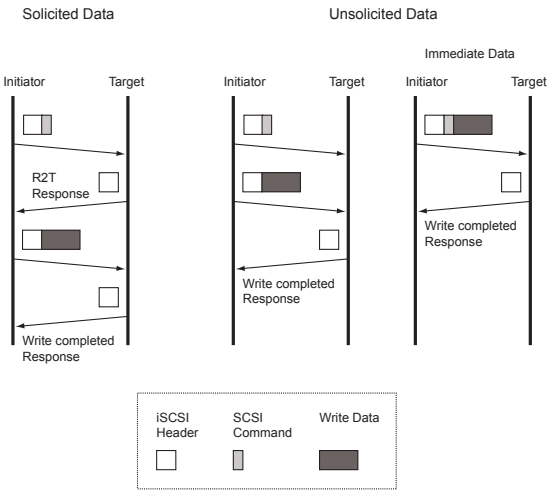

図 3 iSCSI プロトコルの書き込み命令の分類
Fig. 3 Data transfer overview.

式を**図 3** にまとめた．

Solicited Data 方式は，ターゲットがデータを受け取るタイミングを制御できるため，ターゲットでのプロトコル処理が容易であり，ディスクのハードウェア特性に合わせた順序でイニシエータにデータを要求する等の性能向上手法が利用できる．一方，Unsolicited Data 方式は，Solicited Data 方式よりも必要なパケットの数が少ないため，プロトコル処理のオーバヘッドと遅延が小さく，効率が良いという利点を持つ．

3.4.1 Solicited Data 方式

UNH ターゲットと Threaded-Ardis ターゲットは，Solicited Data 方式の WRITE コマンドを受け取ると，書き込みデータを保持するための一時バッファを確保し，R2T コマンドをイニシエータに送信する．READ スレッドは，イニシエータから送られてきた書き込みデータを，先ほど確保した一時バッファにコピーする．READ スレッドがすべての書き込みデータを受け取ると，WRITE コマンドは IO スレッドに渡される．

UNH ファイルモードターゲットと Threaded-Ardis ターゲットの IO スレッドは一時バッファを引数として，カーネル標準ファイルインタフェースを呼び出すことで，一時バッファからページキャッシュへのデータコピー，ページキャッシュからディスクへの書き込みに必要なディスク IO 命令を実行できる．

一方，UNH ディスクモードターゲットの IO スレッドは，ページキャッシュを利用せずに，一時バッファを使って，カーネル標準 SCSI インタフェースを呼び出し，SCSI ディスクへの書き込みを実行する．

Ardis ターゲットは，Solicited Data 方式の WRITE コマンドを受け取ると，後で書き込みデータを受け取った際に，データのコピー先となるページキャッシュをあらかじめ確保，その参照カウンタを増やし☆，R2T コマンド送信する．READ スレッドが，すべての書き込みデータをソケットバッファからページキャッシュにコピーした後，IO スレッドは，更新したページキャッシュのロックを取得し，ブロックデバイスへの書き込みを実行する．

3.4.2 Unsolicited Data 方式

UNH ターゲットと Threaded-Ardis ターゲットは，Solicited Data 方式とまったく同じ方法で，Unsolicited Data 方式の書き込みを処理する．一方，Ardis ターゲットは，Solicited Data 方式とは異なる方法で，Unsolicited Data 方式の書き込みを処理する．

Ardis ターゲットが Solicited Data 方式で用いていたソケットバッファからページキャッシュに直接データをコピーする方法で，Unsolicited Data 方式の書き込みを処理すると，対象となるページキャッシュを確保するまでに時間がかかる場合に性能の低下を招く．ページキャッシュを確保するまで，書き込みデータをソケットバッファから取り除くことができないので，ターゲットは次の PDU の処理を始めることができない．

上記の問題を避けるために，Ardis ターゲットは UNH ターゲットや Threaded-Ardis ターゲットと同じ方法で，Unsolicited Data 方式の書き込みを処理を実行する．つまり，書き込みデータはソケットバッファから一時バッファを経由し，ページキャッシュにコピーされる．ただし，対応するディスクブロックがページキャッシュに保持されておらず，新たにページキャッシュを割り当てる必要がある場合は，一時バッファからページキャッシュへのデータコピーを回避することができる．この操作は，一時バッファとして確保したメモリをページキャッシュとして再利用することで実現されているが，仮想記憶システムへの変更を必要とする．

図 4 に，Solicited Data 方式と Unsolicited Data 方式の WRITE コマンド処理におけるデータの流れを示す．破線は，条件によっては，回避できるデータコピーを示している．

3.5 読み込み処理

UNH ターゲットは，READ コマンドを受け取ると，要求されたデータを保存できる大きさの一時バッファ

☆ 確保したページがディスクの内容を保持しておらず，さらにそのページキャッシュのロックを取得できなかった場合，一時バッファを経由した書き込み方法を利用する．

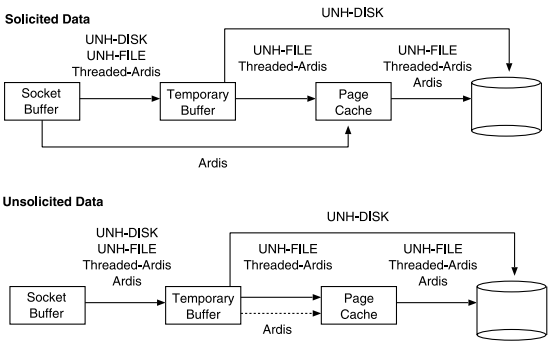

図 4 書き込み処理でのデータパス
Fig. 4 Write data path.

を確保する．

UNH ファイルモードターゲットの IO スレッドは，一時バッファへのポインタとデータ長を指定し，カーネル標準ファイルインタフェースを呼び出すことで，ディスクから一時バッファにデータを読み込む．カーネル標準ファイルインタフェースは，必要なデータがすでにページキャッシュに保持されていれば，ページキャッシュから一時バッファにデータをコピーする．もし，データがページキャッシュに保持されていなければ，ディスクから新しく割り当てたページキャッシュへの読み込みが完了してから，データを一時バッファにコピーする．

UNH ディスクモードターゲットの IO スレッドは，ページキャッシュを利用せずに，カーネル標準 SCSI インタフェースを用いて，つねにディスク IO 命令を発行し，SCSI ディスクから一時バッファにデータを読み込む．

Ardis ターゲットは，READ 命令を受け取り，必要なデータがページキャッシュにすでに保持されていれば，ページキャッシュの参照カウンタを増やす．もし，データがページキャッシュに見つからなければ，新しいページキャッシュを割り当て，読み込みディスク IO 命令を開始する．

Threaded-Ardis ターゲットも，Ardis ターゲット同様に，必要なページキャッシュの参照カウンタを増やす．しかし，Ardis ターゲットとは異なり，カーネルへの変更や独自の関数を必要とせず，カーネル標準ファイルインタフェースを用いて，上記の操作を実装している．Threaded-Ardis ターゲットの IO スレッドは，スリープすることが可能な設計であるため，カーネル標準ファイルインタフェースを用いて，上記の機能を実装できる．

UNH ターゲットは，一時バッファに送信するデータが準備できると，WRITE スレッドが，一時バッファからソケットバッファへデータをコピーすることで，イニシエータに送信する．

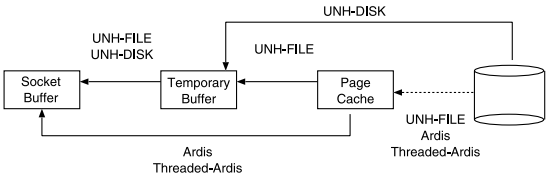

図 5 読み込み処理でのデータパス
Fig. 5 Read data path.

Ardis ターゲットと Threaded-Ardis ターゲットは，カーネル標準インタフェースの sendpage 関数を利用して，ページキャッシュのデータをソケットバッファにコピーすることなく，イニシエータに送信する．

図 5 に，READ コマンド処理におけるデータの流れを示す．破線は，条件によっては，回避できるデータコピーを示している．

4. 性能評価

4.1 環境

ターゲットとして用いた計算機は，Xeon 2.8 GHz に 2 GB のメモリを搭載し，Linux カーネルのバージョン 2.4.25 が動作する．LSI Logic 社の 53C1030 チップを搭載した Ultra320 SCSI ホストバスアダプタにつながれた，Maxtor 社の Atlas 10 K SCSI ドライブ（36.7 GB 10,000 RPM）を，イニシエータに提供するディスクイメージとして用いた．UNH ファイルモードターゲットの場合は，ext2 ファイルシステムを構築し，作成したファイルをイニシエータに提供するディスクイメージとして用いた．

イニシエータとして用いた計算機は，Xeon 2 GHz に 1 GB のメモリを搭載し，Linux カーネルのバージョン 2.6.4 が動作する．イニシエータソフトウェアは，Cisco 社のオープンソースソフトウェア[12] のバージョン 4.0.1.1 を使った．

イニシエータとターゲットは，ともに Intel Pro/1000 MT Server Adapter を装着し，Extreme 社の Summit 7i Gigabit Ethernet スイッチを使って接続されている．

最良の書き込み性能が得られるように，iSCSI プロトコルのパラメータを**表 2** のように設定し，Unsolicited Data 方式の Immediate Data 型の書き込みが実行されるようにした．

ネットワークサーバベンチマークを除いて，コールドキャッシュの状態から実験を開始した．結果は 5 回の測定の平均値である．

表 2 iSCSI パラメータ設定
Table 2 iSCSI parameters.

項目	値
InitialR2T	On
ImmediateData	On
MaxRecvDataSegmentLength	128 KB
MaxBurstLength	256 KB
FirstBurstLength	64 KB
MaxConnections	1
MaxOutstandingR2T	1

表 3 評価した iSCSI ターゲットソフトウェア
Table 3 System comparison.

	特徴
Ardis	Ardis ターゲット
Threaded-Ardis	我々の実装したターゲット
UNH-disk	ディスクモードの IO スレッドを用いる標準の UNH ターゲット
UNH-file	ファイルモードの IO スレッドを用いる標準の UNH ターゲット
UNH-file-sync	WRITE コマンドの完了通知をデータとディスクを同期してから送るように変更した UNH ファイルモードターゲット

4.2 評価したターゲットソフトウェア

3 章で設計を比較した 4 種類に，UNH ファイルモードターゲットを変更した実装（UNH-file-sync）を加えた，合計 5 種類の iSCSI ターゲットソフトウェアの性能を評価した．**表 3** に一覧を示す．

UNH-file-sync は，DAS 環境との互換性を実現するため，sys_fsync() 関数を用いて，WRITE コマンドの処理の際に，書き込みデータをディスクと同期してから，イニシエータに完了通知を送信するように変更した UNH-file の実装である．なお，UNH-file-sync と UNH-file との違いは，WRITE コマンドの処理方法だけであるため，読み込み負荷における，UNH-file-sync の性能測定は省略した．

4.3 マイクロベンチマーク

我々が実装したマイクロベンチマークをイニシエータで動作させて，各ターゲットの基本的な性能上の特徴を調査した．このベンチマークはカーネル内部で動作し，ブロック I/O レイヤに直接アクセスすることで，Linux カーネルの IO 命令最適化機能☆の影響を受けず，指定した大きさと回数の IO 要求を正確に生成できる利点を持つ．我々は，SCSI コマンドが要求する IO サイズを 2 KB〜32 KB に設定し，それぞれ，50,000 回のシーケンシャル，ランダムなディスク IO 命令を発生させた．

☆ ページキャッシュ，連続するディスクブロックに対する複数のディスク IO 命令をマージする機能等を指す．

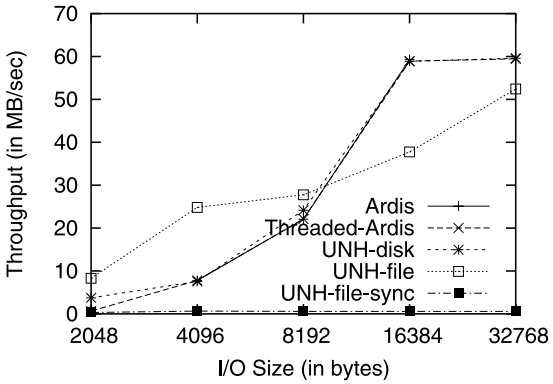

図 6 シーケンシャルな書き込み性能
Fig. 6 Sequential write results.

Ardis ターゲットに関しては，そのバグのため，IO サイズが 2 KB のシーケンシャルな書き込み性能，32 KB ではすべての条件での性能が測定できなかった．

Ardis ターゲットは，つねにページサイズ，x86 アーキテクチャでは 4 KB の単位でディスク入出力を実行するため，ページサイズよりも小さいサイズの書き込み要求では，最初にディスクから 4 KB のデータを読み出し，その一部をイニシエータから送られてきたデータで更新し，4 KB のデータで再びディスクと同期する操作を行う．同一のページに対して，そのような要求が連続した場合にシステムが停止するバグがあり，シーケンシャルな 2 KB の書き込み性能が測定できなかった．また，32 KB を超える IO 要求処理にもバグがあり，読み書き両方の性能が測定できなかった．

なお，Threaded-Ardis ターゲットは，Ardis ターゲットの一部のコードを利用しているが，ディスク入出力部分はカーネル標準ファイルインタフェースを使った異なる実装であるため，上記のバグは存在せず，すべての条件で測定を行っている．

シーケンシャルな書き込み性能の測定結果を**図 6** に示した．事前に予想したように，UNH-file-sync の性能は，他の 4 つのターゲットの性能と比較して，1%〜5%程度と非常に低い．SCSI コマンドを並列処理できないことに加えて，sys_fsync 関数によるページキャッシュとディスクの同期操作のオーバヘッドが，性能を大きく低下させることが確認できる．

UNH-file-sync と同様に，SCSI コマンドを並列処理できない UNH-file が高速な理由は，3 章で述べたように，書き込みデータとディスクを同期することに関して何も保証もしないためである．特に，IO サイズが 8 KB 以下の条件では，ベンチマーク実行中に，ターゲットはディスク IO 命令をまったく実行しない

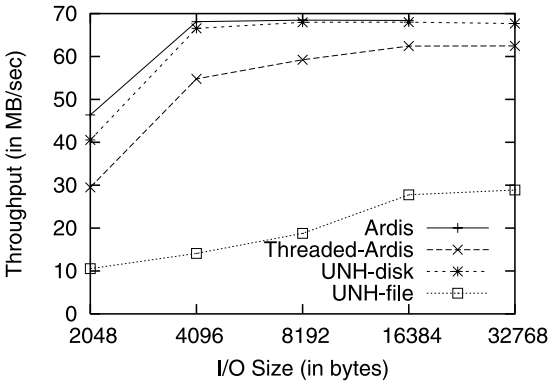

図 7 シーケンシャルな読み込み性能
Fig. 7 Sequential read results.

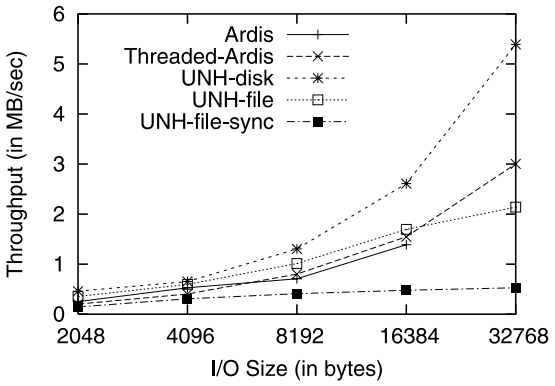

図 8 ランダムな書き込み性能
Fig. 8 Random write results.

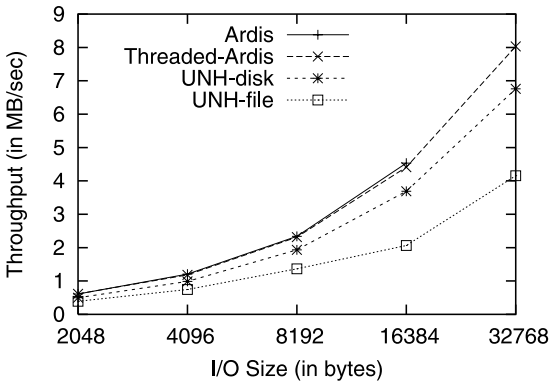

図 9 ランダムな読み込み性能
Fig. 9 Random read results.

ため，他のターゲットよりも高速である．

カーネルバージョン 2.4.25 と UNH-file が利用した ext2 ファイルシステムの組合せは，(1) 更新されたページキャッシュの量がシステム全体のページキャッシュ量の 30%を超える，(2) ページキャッシュが更新されてから 30 秒以上経過する，のうちどちらかの条件が満たされると，更新されたページキャッシュのディスクへの書き込みを開始する．IO サイズが 8 KB 以下の場合は，いずれの条件も満さないので，更新されたページキャッシュのディスクへの書き込みは発生しない．

Ardis，Threaded-Ardis，UNH-disk の 3 つのターゲットから，ほぼ同じ性能が得られている．Threaded-Ardis における，IO スレッドのマルチスレッド化やカーネル標準ファイルインタフェースを利用するオーバヘッドは小さいことが確認できた．

図 7 はシーケンシャルな読み込み性能の測定結果である．予想どおり，SCSI コマンドを並列処理できない UNH-file の性能が最も低い．しかし，カーネル標準ファイルインタフェースが内部で実行する先読みと呼ばれる，ファイルを読み込む際に，要求された位置よりも先のデータのディスク IO 命令を投機的に開始する機能のため，最悪のケースでも，UNH-file の性能は，他のターゲットの 20%程度となっている．

最も性能が高かったのは，Ardis である．しかし，IO サイズが 2 KB のときを除けば，UNH-disk の性能は Ardis の性能とほぼ等しい．2 KB で，Ardis が UNH-disk よりも高性能なのは，Ardis がつねにページキャッシュの大きさ（4 KB）でディスク IO 命令を実行するためである．その結果，前述の先読み機能と同様の効果が得られ，実際のディスク IO の実行回数を削減することができる．

Threaded-Ardis は，Ardis の性能の 63〜91%の性能となった．この性能低下は，Ardis が用いる独自に実装したインタフェースと，Threaded-Ardis が用いたカーネル標準ファイルインタフェースの性能差によるものである．カーネル標準ファイルインタフェースは，先読み機能等，Ardis の独自インタフェースが実行するページキャッシュ操作とディスク IO 命令以外の作業を行うため，このような性能差が生じる．Threaded-Ardis の READ コマンド処理を，カーネル標準ファイルインタフェースを用いる方法から，Ardis と同様にページキャッシュ操作とディスク IO 命令を直接実行する方法に変更して，性能を測定したところ，Ardis との性能差は 2%未満になった．その結果は，WRITE コマンド処理同様，マルチスレッド化による性能低下は非常に小さいことを意味する．

図 8，**図 9** は，ランダムな書き込み，読み込み性能の測定結果である．いずれの場合も，ランダムアクセスでは，ディスクドライブ性能から得られる性能が低く，各ターゲットの性能は，シーケンシャルな負荷の 10 分の 1 程度に低下している．

ランダムアクセスの測定結果は，シーケンシャルアクセスの結果から予測できる結果から大きく外れてはいない．

例外の 1 つとして，ランダムアクセスのようにターゲットが同時に処理する SCSI コマンド数が多くなる負荷の処理に欠点を持つ Ardis の性能が低い点があげられる．ページキャッシュ操作のインタフェースが非同期 IO 操作でないため，Ardis は，ページキャッシュの操作完了を待ち合わせて，起床した後に，操作が終了したページキャッシュを見つけるために，IO スレッドが処理中のページキャッシュすべてを検索しなければならない．したがって，完了通知を送信するまでに必要になる時間の長さは，処理中の SCSI コマンドの数に比例して長くなる．

4.4 マクロベンチマーク

実環境で発生する負荷に対する各ターゲットの性能を測定するために，我々は 2 種類のマクロベンチマークを実行した．

マクロベンチマークでは，商用環境における，オープンソースと汎用のハードウェアを用いて実装されたストレージシステムの有効性を確認するため，エントリクラスの商用 iSCSI ストレージシステムを評価対象に加えた．このストレージシステムを構成するハードウェア，ソフトウェアに関する情報は公開されていない．

4.4.1 Postmark ベンチマーク

Postmark ベンチマーク[13] は，インターネットサービスプロバイダのメールサーバやニュースサーバで発生する負荷を模擬するベンチマークであり，大量の小さなサイズの短命なファイルを操作する負荷を生成する．Postmark ベンチマークの負荷は書き込みが支配的であり，我々の測定では，イニシエータがターゲットに要求したディスク IO 命令の 99%以上を書き込みが占めていた．

我々は，サイズが 512 B から 16 KB のファイル，20,000 個を使い，50,000 回のトランザクションを実行した．イニシエータはターゲットが提供するディスク上にブロックサイズが 4 KB の ext2 ファイルシステムを構築し，ベンチマークを実行した．

表 4 がベンチマークの結果である．予想どおり，UNH-file-sync の性能が最も悪く，最も性能が良かった商用システムの性能の 4 分の 1 以下であった．

Ardis，Threaded-Ardis，UNH-disk，商用システムの性能はほぼ等しく，それらの性能差は 3%未満であった．

表 4 Postmark ベンチマークの結果
Table 4 Postmark results.

	Throughput (transactions per second)
Ardis	641.0
Threaded-Ardis	634.4
UNH-disk	644.0
UNH-file	527.0
UNH-file-sync	154.8
商用システム	649.2

表 5 ネットワークサーバベンチマークの結果
Table 5 Network server benchmark results.

	Throughput (MB/sec)	
ファイルの個数	512	1024
Ardis	303.3	47.1
Threaded-Ardis	294.8	43.5
UNH-disk	117.8	32.3
UNH-file	293.3	—
商用システム	239.3	78.8
ターゲットでのページキャッシュヒット率	85.4 %	25.4 %

4.4.2 ネットワークサーバベンチマーク

HTTP サーバや FTP サーバ用ストレージシステムで発生する，読み込みが支配的な負荷に対する性能を測定するために，複数のファイルをランダムな順番で読み込むベンチマークを実行した．このベンチマークは，非同期 IO を用いて，1 つのプロセスが複数ファイルの読み込みを並列に実行できるように実装されている．

イニシエータは，ターゲットが提供するディスク上に，ブロックサイズ 4 KB の ext2 ファイルシステムを構築し，1 つのディレクトリにサイズが 2 MB がファイルを 512 個を作成した．ベンチマークは，ディレクトリ内のファイルの中から，1 つをランダムに選択して読み込む操作を 4,096 回繰り返す．また，ファイル数を 1,024 個にした場合の性能も測定した．なお，UNH-file は，イニシエータに提供できるディスクの最大サイズが 2 GB であるため，ファイル数が 1,024 個の条件での性能は測定できない．

このベンチマークは，ターゲットのページキャッシュが性能に与える影響を測定することが目的であるため，イニシエータ，ターゲットともにホットキャッシュの状態から測定を開始した．

表 5 に測定結果を示した．スループットに加えて，ターゲットの性能とページキャッシュの関係を解析するために，ターゲットでのページキャッシュヒット率も測定した．この数値は，イニシエーターから要求された読み込み命令のうち，必要なデータがページキャッ

シュに保持されており，ターゲットが実際のディスクIO 命令を回避できた割合を示す．

ファイルが 512 個の条件では，ターゲットでのページキャッシュのヒット率が高く，その影響が性能に大きく影響していることが分かる．ページキャッシュを利用する，Ardis，Threaded-Ardis，UNH-file ターゲットからは，同じような性能が得られているが，ページキャッシュを利用せずに，すべての SCSI コマンドがディスク IO 命令を実行する UNH-disk の性能は，それらのターゲットの性能の 3 分の 1 程度になっている．

ファイルが 1024 個の条件では，対象となるデータセットの大きさが 2 GB を超え，ターゲットのページキャッシュヒット率は 25.4%まで下がった．したがって，ページキャッシュが性能に与える影響も小さくなり，UNH-disk の性能は，Arids の性能の 68.6%まで改善した．

ターゲットが 2 GB のメモリを搭載し，ファイルの合計サイズが 2 GB であるにもかかわらず，ページキャッシュヒット率が 25.4%と低いのは，Linux カーネルが，896 MB 以降のメモリをブロックデバイスのページキャッシュとして利用しないためである．

商用システムの性能は，ファイルが 512 個の条件では，ページキャッシュを利用する 3 種類のターゲットの性能の 8 割程度である．逆に，ファイルが 1,024 個の場合，商用システムの方が性能が良い．商用システムは，ディスクブロックをキャッシュする機構を利用しており，ファイルが 1,024 個の場合でも，キャッシュヒット率を高く保つことができる量のメモリが使われている，と考えられる．

4.5 議　　論

読み込みに関しては，カーネル標準ファイルインタフェースと独自に実装した iSCSI に最適化した関数の性能差が見られた．その理由は，カーネル標準ファイルインタフェースの先読み操作によるオーバヘッド増加のためと考えている．ディスクの指定した範囲を読み込む操作のみを実行する高レベルなカーネル標準インタフェースとして考えられるのは，カーネルバージョン 2.6 に導入された asynchronous I/O（非同期 I/O）がある．しかし，asynchronous I/O の実装は，現在も活発に開発が進められている段階で，十分に安定しているとはいえない．今後の評価が必要である．

Threaded-Ardis は，sys_fsync 関数の代わりに，指定した範囲のダーティなページキャッシュだけをディスクと同期する機能を低レベルなカーネル標準関数を利用して実装している．この関数の大きさは，40 行以下と非常に小さいが，高レベルなカーネル標準インタフェースで置き換えることができれば，さらにターゲットの設計を簡略化できる．上記の asynchronous I/O インタフェースには，指定した範囲のダーティなページキャッシュだけをディスクと同期する機能が実装される予定となっており，この機能に関しても今後の評価が必要である．

x86 アーキテクチャの場合，Linux カーネルは，896 MB 以降のメモリをブロックデバイスのページキャッシュとして利用できない制限があり，読み込みが支配的な負荷で，性能の大きな低下が見られた．この制限は，カーネルバージョン 2.6 系でも解決されておらず，改善が必要である．なお，AMD 社の Opteron 等，64 ビットのアーキテクチャでは，この制限はない．

5. ま と め

本稿では，2 種類のオープンソースソフトウェアを用いて，iSCSI ターゲットシステムの設計技法とその性能との関係について議論した．まず，iSCSI ターゲットがオペレーティングシステムに要求する機能を明らかにし，性能測定によって，汎用的なオペレーティングシステムのカーネルが提供する標準的な機能だけを使った実装方法と，カーネルに変更を加えて実装した iSCSI ターゲットに最適化された機能を使った実装方法の性能差を明らかにした．その結果，既存の前者のアプローチでは，書き込みが支配的な負荷，読み込みが支配的な負荷，2 種類の実環境負荷を模擬するベンチマークの両方に対して，後者と同様の性能を実現することが困難であることが分かった．しかし，複数のスレッドを利用してディスク IO 命令を実行する設計に変更することで，カーネルに変更を加えることなく，カーネルの標準的な機能を利用して，後者の最適化された機能を用いる実装と同等の性能を達成可能であることを示した．また，オープンソースと汎用ハードウェアを用いて実現された iSCSI ターゲットシステムが，エントリクラスの商用 iSCSI ターゲットシステムと同等の性能を実現できることを確認した．

Threaded-Ardis のソースコードは
http://iscsitarget.sourceforge.net/
で入手可能である．

参 考 文 献

1) Satran, J., Meth, K., Sapuntzakis, C., Chadalapaka, M. and Zeidner, E.: Internet Small Computer Systems Interface, RFC 3720, (iSCSI) (2004).

2) UNH-iSCSI Initiator and Target (2003). http://unh-iscsi.sourceforge.net/
3) Ardis Technologies iSCSI target implementation: (2003). http://www.ardistech.com/iscsi/
4) Radkov, P., Yin, L., Goyal, P. and Sarkar, P.: A Performance Comparison of NFS and iSCSI for IP-Networked Storage, *USENIX Conference on File and Storage Technologies*, San Francisco, CA, pp.101–114 (2004).
5) Aiken, S., Grunwald, D., Pleszkun, A.R. and Willek, J.: A Performance Analysis of the iSCSI Protocol, *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies*, San Diego, California, pp.123–134 (2003).
6) Sarkar, P., Uttamchandani, S. and Voruganti, K.: Storage over IP: When Does Hardware Support Help?, *Conference on File and Storage Technologies* (*FAST 03*), San Francisco, CA, USENIX, pp.231–244 (2003).
7) Ng, W.T., Hillyer, B., Shriver, E., Gabber, E. and Özden, B.: Obtaining High Performance for Storage Outsourcing, *USENIX Conference on File and Storage Technologies*, Monterey, CA, pp.145–158 (2002).
8) Palekar, A., Ganapathy, N., Chadda, A. and Russell, R.D.: Design and implementation of a Linux SCSI target for storage area networks, *5th Annual Linux Showcase & Conference*, Atlanta, GA, USENIX (2001).
9) Intel iSCSI Reference Implementation (2001). http://sourceforge.net/projects/intel-iscsi/
10) Teigland, D. and Mauelshagen, H.: Volume Managers in Linux, *USENIX Annual Technical Conference*, Boston, MA, pp.185–198 (2001).
11) Seltzer, M., Ganger, G., McKusick, M.K., Smith, K., Soules, C. and Stein, C.: Journaling Versus Soft Updates: Asynchronous Meta-data Protection in File Systems, *USENIX Annual Technical Conference*, San Diego, CA, pp.71–84 (2000).
12) Cisco iSCSI initiator Implementation (2001). http://linux-iscsi.sourceforge.net/
13) Katcher, J.: PostMark: A New File System Benchmark, Technical Report TR3022, Network Appliance (1997).

(平成 16 年 5 月 13 日受付)
(平成 16 年 9 月 16 日採録)

藤田　智成（正会員）

2000 年早稲田大学大学院理工学研究科修士課程修了．同年日本電信電話株式会社入社．オペレーティングシステムに関する研究に従事．ACM，USENIX 各会員．

小河原成哲

1994 年慶應義塾大学理工学研究科電気工学専攻修士課程修了．同年日本電信電話株式会社入社．波長多重交換システムの研究に従事．現在，NTT サイバーソリューション研究所研究主任．1999 年電子情報通信学会論文賞受賞．電子情報通信学会会員．