社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－OS－99（21）
2005／5／27

iSCSIイニシエータの設計についての一考察

藤田智成[†]　矢田浩二[†]

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，SCSI コマンドを発行する計算機（イニシエータ）の設計に関する議論はされていない．本稿では，オープンソースの iSCSI イニシエータソフトウェアの設計手法を比較し，性能，信頼性について論じる．実験では，1 Gbps，10 Gbps イーサネット環境の両方で，基本的な設計に基づいたイニシエータと，メモリコピー回避，TCP/IP スタック内部での iSCSI プロトコル処理等，複数の高性能化手法を用いたイニシエータのシーケンシャルリード性能を評価した．その結果，設計の違いによる性能差は確認できなかった．

Study on iSCSI initiator designs

Tomonori Fujita[†] and Kouji Yata[†]

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, the design of initiators, issuing SCSI commands, have not been investigated. This paper analyzes the design of open-source iSCSI initiator software for a general purpose operating system. Our experiments show that a straightforward iSCSI initiator and an iSCSI initiator using techniques for high performance such as zero-copy and integrating iSCSI protocol processing in the TCP/IP stack, provide comparable throughputs in both 1 Gigabit Ethernet and 10 Gigabit Ethernet.

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，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た，設計手法の妥当性に関する議論はされていない．

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である，Linux で動作し，iSCSI イニシエータ機能を実現するオープンソースソフトウェアの設計手法を解析した．対象にした Linux カーネルのバージョンは，2.6.12-rc2 である．

iSCSI イニシエータ機能は，ネットワークアダプタのハードウェアに実装することもできる．ソフトウェアで iSCSI イニシエータ機能を実現する手法よりも，ハードウェアによる実装は性能面で有利である．しかし，ソフトウェアによる実装は，通常の安価な Ethernet NIC を利用できるため，コスト面で有利である．

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

また，機能拡張，バグ修正が容易であるという利点もある．

本稿の構成は以下の通りである．まず，2 章で，Linux の SCSI サブシステムと iSCSI イニシエータについて説明し，3 章で，イニシエータソフトウェアの設計を解析する．4 章で，性能評価結果を示す．5 章で，イニシエータの設計手法について議論し，6 章で関連研究にについて述べる．最後に，7 章でまとめる．

2. Linux における SCSI サブシステム

Linux カーネルでは，システムに接続された SCSI ホストバスアダプタ（HBA）のドライバを管理するカーネルサブシステムを SCSI ミドルレイヤと呼ぶ．SCSI ミドルレイヤは，SCSI HBA とブロック I/O レイヤ等，カーネル内の様々なカーネルサブシステムを接続する役割を果たす．ブロック I/O レイヤは，SCSI, IDE などの様々な物理インターフェイスのデバイスにアクセスするための，統一されたインターフェイスを提供するカーネルサブシステムである．

SCSI HBA のデバイスドライバは，LLD（Low Level Driver），または，LLDD（Low Level Device Driver）と呼ばれる．SCSI ミドルレイヤから見ると，iSCSI イニシエータ機能は，HBA の一種であり，iSCSI イニシエータソフトウェアは，LLD に相当する．

2.1 Scsi_Host と scsi_host_template 構造体

SCSI ミドルレイヤにおけるホストとは，計算機の物理バスと SCSI イニシエータのポートの組を意味する．SCSI イニシエータのポートは，SCSI ターゲットと接続され，SCSI コマンドを送信する．ターゲットは複数のデバイス（ディスク等）を持つことができる．

通常の（ハードウェア）SCSI HBA と異なり，iSCSI イニシエータは仮想的な HBA であるため，ホストは計算機の物理バスとは関係なく，イニシエータとターゲット間のセッションに対応する．セッションは，イニシエータとターゲット間の通信のために使われる TCP/IP コネクションを保持している．

SCSI ミドルレイヤは，ホストごとに，Scsi_Host 構造体を割り当てる．

scsi_host_template は，LLD がその動作を定義するために使う構造体（同時に処理できる SCSI コマンド数等のデータを保持する）で，LLD と SCSI ミドルレイヤ間の最も基本的なインタフェースである．Scsi_Host 構造体は，対応する LLD の scsi_host_template へのポインタを保持する．

LLD は，scsi_host_template 構造体のデータのうち，SCSI コマンドをホストにキューイングするための関数へのポインタである queuecommand，エラー処理に関する複数の関数へのポインタを必ず定義する必要がある．LLD が定義していない，scsi_host_template 構造体内のデータについては，SCSI ミドルレイヤの標準の値が使われる．

2.2 queuecommand 関数

SCSI コマンドの管理には，scsi_cmnd 構造体が使われる．SCSI ミドルレイヤは，ホストに SCSI コマンドをキューイングするために，scsi_cmnd 構造体へのポインタと，SCSI コマンドの完了を SCSI ミドルレイヤに通知するために LLD が使う，done 関数へのポインタを引数に，scsi_host_template 構造体の queuecommand ポインタが指す関数を呼ぶ．

通常の HBA の LLD の場合，queuecommand ポインタが指す関数は，ハードウェアのレジスタを操作し，SCSI コマンドの処理開始を命令する．LLD は，ハードウェア割り込みを利用して，SCSI コマンド処理の完了後に，指定された done 関数を呼び，SCSI ミドルレイヤに SCSI コマンドの完了を通知する．

iSCSI イニシエータドライバの場合，queuecommand ポインタが指す関数は，SCSI コマンドを，iSCSI Protocol Data Unit（PDU）として，TCP/IP コネクションを通じてターゲットに送信する．ターゲットから，SCSI コマンドの完了通知を含む iSCSI PDU を受け取ると，done 関数を呼ぶ．

2.3 エラー処理

SCSI ミドルレイヤは，ホストにキューイングした SCSI コマンドが正常に終了しなかった場合のエラー処理のために，ホストごとにカーネルスレッドを一つ生成する（エラーハンドラスレッドと呼ぶ）．

scsi_host_template 構造体には，エラー処理に使われる 6 種類の関数へのポインタが含まれている．

eh_abort_handler　SCSI コマンドの中止
eh_device_reset_handler　デバイスのリセット
eh_bus_reset_handler　バスのリセット
eh_host_reset_handler　ホストのリセット
eh_strategy_handler　エラーハンドラスレッドが実行する標準のエラー処理を置き換える
eh_timed_out　エラーハンドラスレッドの標準のエラー処理開始前の動作を指定

最初の 4 種類の関数は，エラーハンドラスレッドが，標準のエラー処理の実行中に呼び出す関数で，LLD は，このうち，少なくとも 1 つを定義する必要がある．

SCSI コマンドが一定時間内に完了しない場合（タイムアウト），エラー処理が終了するまで，SCSI サブシステムアは，そのホストへのアクセスを禁止する．1 つの SCSI コマンドがタイムアウトすると，ホストにキューイングされている全ての SCSI コマンドがタイムアウトしてから，エラーハンドラスレッドはエラー処理を開始する．

標準エラー処理は，タイムアウトした SCSI コマンドの実行中止に失敗した場合（デバイスが反応しない等），デバイス，バス，ホストと，徐々に大きな範囲で，エラーの回復を試みる．エラーハンドラスレッドは，SCSI コマンドを実行可能な状態に復旧するまで，

上記の最初の 4 種類の関数のうち，LLD が定義している関数を順番に呼び出す．エラー処理終了後，デバイスが復旧した場合は，タイムアウトした SCSI コマンドは再度実行される．

3. iSCSI イニシエータの設計

3.1 既存 iSCSI イニシエータの概要

現在，Linux で動作する，GPL ライセンスの iSCSI イニシエータソフトウェアは，4 種類存在する．

3.1.1 core-iscsi

Linux を使って，iSCSI ストレージシステム（ターゲット）を実現するソフトウェアを販売する PyX Technologies 社が，一緒に販売していたイニシエータを，2005 年 2 月に公開したものが，core-iscsi[2] である．iSCSI 規格でオプションとなっている機能を数多く実装しており，機能的には，最も進んだイニシエータである．

3.1.2 open-iscsi

open-iscsi[3] は，10 Gigabit Ethernet NIC を販売する Neterion 社に勤めるプログラマが開発を始めたイニシエータである．カーネル空間のコードは新たに書かれたものであるが，ユーザ空間で動作するコードの多くは，後述する sfnet-iscsi をベースにしている．現在，Linux カーネルバージョン 2.6 系統の管理者である，Andrew Morton の開発ツリーにマージされている．

3.1.3 sfnet-iscsi

sfnet-iscsi[4] は，Linux の標準の iSCSI イニシエータソフトウェアとして考えられており，多くの商用の iSCSI ストレージシステム（ターゲット）がサポートしている．元々，Cisco 社が自社のストレージ製品のためのイニシエータとして公開したコードであるが，現在は，Cisco 社以外の，Netapp，IBM 等の大手ストレージベンダーのプログラマも，主要な開発者として参加している．しかし，開発者は，現在のコードの開発を中止し，open-iscsi プロジェクトと共同で．open-iscsi のコードを使った次期バージョンの開発に取り組んでいる．

3.1.4 UNH-iscsi

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-iscsi[5] は，iSCSI イニシエータドライバのリファレンスとして使用されることを目的として実装されており，研究等の目的で広く使われているが，性能，ユーザインタフェース面等で，商用環境を想定した実装がされていない．

3.2 設計の詳細

本稿では，標準の Linux カーネルに取り込まれる可能性が高い，core-iscsi と open-iscsi の設計を解析する．本稿では，core-iscsi のバージョン 1.6.2.0，open-iscsi のバージョン 0.2 を利用した．

3.2.1 core-iscsi

ソケットからのリードを実行する rx カーネルスレッドと，ライトを実行する tx カーネルスレッドは，ソケットインタフェースを利用して，ソケットのリード・ライトを実行する．ここでの，ソケットインタフェースとは，ユーザ空間で動作する，通常のアプリケーションが，ソケットからのリード・ライトのために用いるシステムコールが，カーネル内部で用いるインタフェースを意味する．

core-iscsi のように，カーネルスレッドがソケットインターフェイスを利用して，ネットワーク I/O を処理する設計は，cisco-iscsi や UNH-iscsi とも共通しており，イニシエータの基本的な設計と言える．

channel_req，channel_scan スレッドは，ターゲットとのセッションの確立等，SCSI ミドルレイヤがキューイングする SCSI コマンドの処理とは非同期に発生するイベントを処理する．

queuecommand 内では，SCSI コマンドごとに iscsi_cmd_t 構造体を割り当て，ミドルレイヤから渡された scsi_cmnd 構造体とリンクし，tx スレッドに処理を引き渡す．tx スレッドは，ターゲットにコマンドを送信する．rx スレッドは，ソケットにデータが届くと起床し，SCSI ミドルレイヤに，SCSI コマンドの処理完了を通知する．

iSCSI プロトコルは，レベル 0 から 2 までの，3 つのレベルのエラー回復動作を定義している．数が大きくなるほど，エラー修復の単位が小さくなるが，動作は複雑になる．イニシエータは，レベル 0 を必ず実装する必要があるが，それ以上のエラー回復動作はオプションとなっている．

core-iscsi は，レベル 2 のエラー回復動作を実装している（ただし，ソースが公開されている現在のバージョンはレベル 0 のみをサポート）．eh_timed_out エラー処理関数を定義し，エラーハンドラースレッドを利用せずに，独自のエラー回復処理を実行する．

3.2.2 open-iscsi

open-iscsi は，他のイニシエータソフトウェアと非常に異なる設計方針に基づいている．他のイニシエータは，大半の機能をカーネル内に実装しているが，open-iscsi は，性能が重要でない機能をユーザ空間に実装している．open-iscsi は，iSCSI プロトコルの機能面では未完成な部分が多いが，カーネル空間のコードが小さい点が，Linux カーネルの開発者に評価されている．

open-iscsi は，他のイニシエータと異なり，カーネルスレッドを生成しない．ソケットからのリードはネットワークスタックの一部の機能として実行される．ソケットへのライトは，queuecommand 関数内で実行される．ソケットバッファに空きがない等の理由で，直ぐにソケットにライトできない場合は，ワークキューと呼ばれる Linux カーネルの機能を利用し，後で実行する．ワークキューは，登録された関数をカーネルス

表 1　サーバ構成

CPU	AMD Opteron 2.4 GHz
Memory	5 GB
NIC	Broadcom BCM5704 1 Gbps
	Chelsio T110 10 Gbps （ドライバ T1.1.4）

レッドを使って遅延実行する機能である．

Linux のネットワーク受信処理は，NIC のハードウェア割り込みで処理される部分と，softirqs と呼ばれる，後で処理される部分に分けられる☆．open-iscsi のソケットからのリードと受信した iSCSI PDU の解釈は，後者の softirqs で実行される．これは，ターゲットとのコネクションが確立した際に，sock 構造体内にある，関数へのポインタである sk_data_ready を置き換えることで実現されている．

エラー処理の一部や，イニシエータとターゲット間で定期的に実行される iSCSI プロトコルによる状態確認コマンド（NOP コマンド）の処理は，ユーザ空間で動作する iscsid デーモンが実行する．iscsid デーモンとカーネル間の非同期イベント通知インタフェースは，netlink 機構を利用している．

open-iscsi は，レベル 0 のエラー回復動作のみを実装している．eh_abort_handler 関数を定義し，エラーハンドラスレッドを利用して，エラーの回復を試みる．eh_abort_handler は，タイムアウトした iSCSI コマンドに対応する，iSCSI プロトコルの TaskManagement コマンドをターゲットに送信して，コマンドの中止を要求する．

4. 性能測定

ネットワークで接続した 2 台の IBM eServer 325 を iSCSI イニシエータ，ターゲットとして動作させ，イニシエータの性能を測定した．実験で使用したハードウェアについて表 1 にまとめた．

ターゲットは，オープンソースソフトウェアの iSCSI enterprise target (IET)[6] を利用した．IET はメモリモードで動作しており，ディスクアクセスによるオーバヘッドは生じない．

イニシエータとターゲットは，1 G 環境では，Extreme 社の Summit 7i Gigabit Ethernet スイッチを使って接続されており，10 G 環境では，クロス接続されている．

図 1，2 は，イニシエータで，disktest ベンチマーク[7] を用いたシーケンシャルリード性能の測定結果である．disktest ベンチマークは，Direct I/O 機能を利用しており，ページキャッシュの影響は含まれていない．図中には，iSCSI プロトコルによる性能の上限を示すために，netperf ベンチマークを使った，ソケットインターフェイスを利用したネットワーク性能測定結果も一緒に示してある．

☆ 一般的なオペレーティングシステム用語では，それぞれ，トップハーフとボトムハーフと呼ばれる部分に相当する．

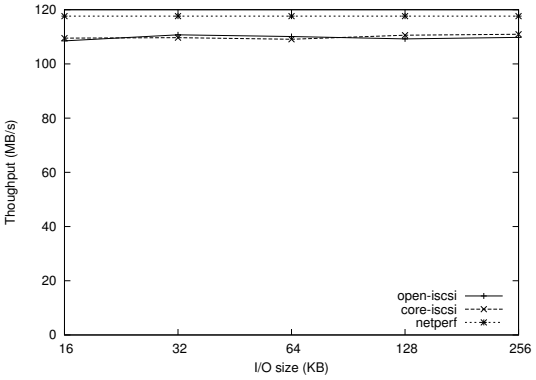

図 1　1G Ethernet でのシーケンシャルリード性能

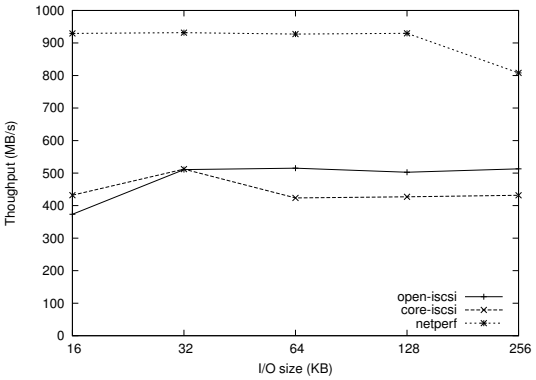

図 2　10G Ethernet でのシーケンシャルリード性能

1 G 環境では，ソケット性能は，物理ネットワークリンク性能の約 94.1%であった．iSCSI プロトコルの性能は，どちらのイニシエータを使った場合も，リンク性能の約 87.8%であった．

10 Gb 環境では，ソケット性能は，最大でも，リンク性能の 74.5%にとどまった．これは，PCI-X バスの性能限界による影響が大きい．iSCSI プロトコルの最大性能は，両イニシエータに大きな差はなく，リンク性能の 41.2%（open-iscsi の 64 KB I/O サイズ時）であった．

5. 議　　論

5.1 性　　能

基本的なイニシエータの設計を用いている core-iscsi と異なり，open-iscsi は複数の高性能化手法を利用している．

第 1 の高速化手法として，open-iscsi は，iSCSI プロトコルの処理をネットワークスタックに統合している．これは，core-iscsi が適用してる，カーネルスレッドとソケットインタフェースを用いてネットワーク I/O

を処理する手法よりも高速である．これは，HTTP サーバ[8] や Linux の NFS サーバ等で用いられている，ネットワーク I/O の典型的な高速化手法である．この手法の欠点は，実装，デバッグが困難になることである．

第 2 の高速化手法として，open-iscsi は，ネットワーク I/O における，冗長なメモリコピーを回避している．ソケットへのライト時は，sendpage インターフェイスを利用することで，メモリコピーを回避している．また，ソケットからのリードでは，ソケットバッファが管理するメモリ領域を直接参照し，iSCSI PDU ヘッダーのメモリコピーを回避している．

第 3 の高速化手法として，open-iscsi は，SCSI ミドルレイヤが呼び出す queuecommand 関数の中でソケットへのライトを実行している．一方，core-iscsi は，tx カーネルスレッドがソケットへのライトを実行するため，queuecommand 関数がキューイングした SCSI コマンドがネットワークスタックへ送られるまで，少なくとも一度はコンテキスト切り替えが必要である．

open-iscsi は上記のような高速化技法を用いているが，今回の実験では，core-iscsi とほぼ同等の性能しか得られなかった．その理由は，core-iscsi が用いている，基本的なイニシエータ設計でもソケット性能の上限を達成可能であるためだと思われる．

10G 環境では，何らかのボトルネックのため，iSCSI プロトコルの性能が，ソケット性能の上限よりも，かなり低い（ソケット性能の約 55.3%）．ボトルネックになりうるカーネルコンポーネントとしては，イニシエータ，SCSI ミドルレイヤー，ブロック I/O レイヤ，Direct I/O インターフェイス，等が考えられる．イニシエータ以外のコンポーネントが性能上のボトルネックになっている場合は，それらを改善すれば，イニシエータの設計が性能に影響をおよぼす可能性もある．

5.2 信　頼　性

5.2.1 ネットワークスタック

iSCSI イニシエータ機能をソフトウェアで実装する手法を用いた場合，高信頼なシステムの実現は容易ではない．既存のイニシエータ実装方法では，オペレーティングシステムの利用可能なメモリ（空きメモリ）の量が極端に不足する状況になると，システムのロックやデータの損失が発生する．これは，Linux カーネル特有の問題ではなく，現在の一般的なオペレーティングシステムに共通する問題である．

オペレーティングシステムは，空きメモリが不足すると，ダーティなページキャッシュをディスクに書き込み，解放することで，空きメモリを確保する．したがって，I/O パス，つまり，I/O デバイスのリード・ライトに関係するソフトウェアコンポーネントは，どんな状況でも，ブロックすることなく，少なくとも一つのライト命令を常に実行することが可能でなければならない．

ブロックする可能性のある I/O パスの例として，ライト命令の処理にために，新たにメモリをアロケートする必要がある I/O パスを考える．この場合，メモリアロケーションに循環依存関係が発生するため，空きメモリが不足する状況で，システムがロックする可能性がある．つまり，空きメモリを確保するためには，ディスクへの書き込みが必要であるが，ディスクへの書き込み処理のために新たにメモリが必要となる状況が発生する．システムが既に空きメモリ不足状態になっているため，I/O パスは新たなメモリのアロケーションができず，ディスクへの書き込みに失敗し，空きメモリが確保できないために，システムはロックする．

上記のような状況では，ダーティなページキャッシュをディスクに書き込まずに，解放する方法以外に，空きメモリを確保する方法はない．しかし，これはアプリケーション等が更新したデータが失われることを意味する．

大量のダーティなページキャッシュによって，空きメモリが極端に不足する状況は，例えば，プロセスが mmap システムコールでファイルをメモリマッピングし，メモリを変更する等の操作で簡単に発生する．

一般に，I/O パスは，ライト命令に関する情報の管理等に，メモリを必要とするため，メモリをアロケートせずに，ライト命令の処理を実現することは現実的ではない．そこで，I/O パスは，あらかじめ，空きメモリが不足する状況にそなえて，占有的に最低限のメモリを確保しておくという手法を用いる．

iSCSI でのライト命令処理は，複数の TCP パケットのターゲットへの送信（書き込み命令とデータ）とターゲットからの受信（完了通知）を必要とする．加えて，パケットの送受信に伴う ACK の処理も必要である．したがって，通常の I/O パス，例えば，ハードウェア HBA の LLD と比較して，I/O パスの要求条件を満たす設計は非常に複雑になる．

現在の TCP スタックや NIC のドライバは，I/O パスとして使われることは想定されておらず，I/O パスの要求条件を満たすように設計されていない．したがって，iSCSI イニシエータを利用すると，システムがロックする可能性がある．

現在，Linux カーネル，iSCSI イニシエータの開発者は，他の I/O パス同様に，パケットの送受信に必要なメモリを iSCSI イニシエータのソケットのためにあらかじめ確保しておく方法を検討している．この方法の問題は，パケットの受信処理が非常に複雑になる点である．NIC は，iSCSI プロトコル以外のパケットも受信するため，NIC のドライバは，空きメモリの不足状況では，あらかじめ確保しておいた受信用パケットのメモリを，iSCSI プロトコルだけに利用し，他のプロトコルのパケットは廃棄する必要がある．その実装方法としては，NIC ドライバに，パケットの中身を理

解する機能を追加する等が考えられる．

5.2.2 ユーザ空間で動作する機能

open-iscsi は，一定時間内に処理することが要求されるイニシエータ機能の一部をユーザ空間で動作する iscsid デーモンが実現してるため，幾つか固有の問題が生じる．

まず，iscsid デーモンは，スケジューリングに関する保証を必要とする．現在は，mlock システムコールを利用し，プロセス空間のページアウトを防ぐだけであるが．今後，リアルタイムスケジューリング機能を利用することが必要だと思われる．

また，標準的なライブラリ関数の幾つかはブロックする可能性があるため（例えば，syslog 関数），iscsid デーモンは利用することができない．この点に関しても，現在の実装は不十分なため，改善が必要である．

6. 関 連 研 究

文献 9) において，Peter 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 の性能比較をおこなっている．また，文献 10) において，Stephen 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 11) において，Julian 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 12) で，Wee Teck 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 13) で，Ashish Palekar らは，UNH イニシエータの設計について説明しているが，他のイニシエータとの比較はしていない．

文献 14) は，Linux で動作し，iSCSI ターゲット機能を実現するオープンソースソフトウェアの設計手法を評価している．

7. ま と め

本稿では，オープンソースソフトウェアを用いて，iSCSI イニシエータの設計技法について，性能，信頼性の面から議論した．基本的な設計手法を用いたイニシエータと，複数の性能向上手法を適用した設計を用いたイニシエータのシーケンシャルリード性能を評価した結果，1 G イーサネット環境では，両イニシエータともにネットワークリンクスピードの 87.8%の性能を記録した．しかし，10 G 環境では，オペレーティングシステムのサブシステムが性能上のボトルネックとなり，両イニシエータともに，リンクスピードの 41.2%の性能しか得られなかった．

今後は，10 G 環境での性能向上手法を検討する予定である．

参 考 文 献

1) Satran, J., Meth, K., Sapuntzakis, C., Chadalapaka, M. and Zeidner, E.: Internet Small Computer Systems Interface (iSCSI) (2004). RFC 3720.
2) PyX Technologies, I.: iSCSI Initiator Core. http://iscsi-initiator-core.org.
3) Yusupov, D. and Aizman, A.: Open-iSCSI. http://www.open-iscsi.org.
4) Cisco iSCSI initiator software. `http://linux-iscsi.sourceforge.net/`.
5) UNH-iSCSI initiator and target. `http://unh-iscsi.sourceforge.net/`.
6) iSCSI enterprise target software. `http://iscsitarget.sourceforge.net/`.
7) The Linux Test Project test suite. `http://ltp.sourceforge.net/`.
8) Joubert, P., King, R., Neves, R., Russinovich, M. and Tracey, J.: High-Performance Memory-Based Web Servers: Kernel and User-Space Performance, *USENIX Annual Technical Conference*, Boston, MA, pp. 175–188 (2001).
9) Radkov, P., Yin, L., , Goyal, P. and Sarkar, P.: A Performance Comparison of NFS and iSCSI for IP-Networked Storage, *the USENIX Conference on File and Storage Technologies*, San Francisco, CA, pp. 101–114 (2004).
10) Aiken, S., Grunwald, D., Pleszkun, A. R. and Willek, J.: A Performance Analysis of the iSCSI Protocol, *the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies*, San Diego, California, pp. 123–134 (2003).
11) Sarkar, P., Uttamchandani, S. and Voruganti, K.: Storage over IP: When Does Hardware Support Help?, *the Conference on File and Storage Technologies (FAST 03)*, San Francisco, CA, USENIX, pp. 231–244 (2003).
12) Ng, W. T., Hillyer, B., Shriver, E., Gabber, E. and Özden, B.: Obtaining High Performance for Storage Outsourcing, *the USENIX Conference on File and Storage Technologies*, Monterey, CA, pp. 145–158 (2002).
13) 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).
14) 藤田智成, 小河原成哲: iSCSI ターゲットソフトウェアの解析, 情報処理学会誌コンピューティングシステム, Vol. 46, No. SIG 3(ACS 8), pp. 38–50 (2005).