Linuxにおけるストレージシステムフレームワークの実現

藤田 智成†

Linux Target Framework（tgt）は，ストレージターゲットドライバのための新しいフレームワークである．tgt が提供するストレージプロトコルに非依存な API を利用することで，SCSI，AOE，NBD 等の様々な SAN プロトコルのターゲットドライバを簡素化することができる．ストレージプロトコルに依存する機能は，各プロトコルライブラリによって提供される．tgt は，信頼性向上，および，開発・保守を容易にするため，ストレージプロトコルの処理等，主要な機能をユーザ空間に実装している．商用環境を模擬した実験で，tgt はカーネル内部に実装されたストレージターゲットドライバと同等の性能を達成した．

Storage System Framework in Linux

Tomonori Fujita†

Linux Target Framework (tgt) is a new facility that allows large simplifications in any storage target drivers. Because of storage-protocol independence, tgt can be used for various storage area network (SAN) protocols such as SCSI, AOE, and NBD. Protocol dependant features like protocol processing are provided by tgt protocol helper libraries. In order to improve reliability and simplify its development and maintenance, a significant portion of tgt is implemented in user space. Our results show that tgt can provide comparable performance that a storage target driver implemented fully in kernel space under realistic workloads.

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デバイス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）から，計算機とストレージデバイス間を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へ移行している．現在，主流の SAN 技術は，Fibre Channel（FC），Ethernet，InfiniBand（IB）等のインターコネクト技術を用いて，SCSI コマンドを転送する．また，SCSI プロトコルを使わない SAN 技術としては，ATA プロトコルを Ethernet フレームにのせる ATA over Ethernet（AOE）[1] や，TCP/IP コネクション上で独自プロトコルを利用する Network Block Device（NBD）[2] 等がある．

様々なベンダが SAN 用ストレージデバイスを販売しているが，汎用の計算機とオペレーティングシステムを利用することで，ストレージデバイスを実現することも可能である．このようなストレージデバイスは，特殊なハードウェアやオペレーティングシステムを用いる商用ストレージデバイスと比較して，性能面では劣るが，機能を自由に追加変更できる，低コストである等の利点を持つ．

本稿では，筆者らが開発を進めている，Linux カーネルのストレージデバイス機能のための共通のフレームワーク，Linux Target Framework（以降，tgt と呼ぶ）について述べる．tgt の特徴は，プロトコル処理等，大半の機能をユーザ空間で実現することで，信頼性の向上，容易な開発・保守を実現するとともに，従来のカーネル内に実装されたストレージデバイス機能と同等の性能を実現した点である．

本稿の構成は以下のとおりである．2 章でストレージデバイスについて説明してから，3 章で設計，具体的な実装について述べる．4 章では，性能評価結果を示す．5 章で関連研究について言及し，最後に 6 章でまとめる．

2. ストレージデバイス

図 1 に，計算機とストレージデバイスの関係を示した．本稿の説明では，SCSI プロトコルのアーキテクチャを用いるが，SCSI プロトコルを使わない SAN プロトコルにも適用できる．

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

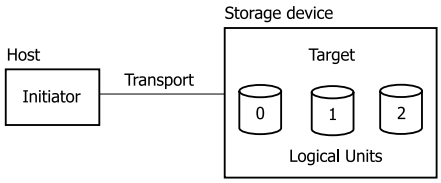


図 1 イニシエータとターゲット
Fig. 1 Initiator and target.

SCSI プロトコルは，クライアントサーバモデルであり，クライアントが要求（SCSI コマンド等）をサーバに送り，サーバが要求を処理し，結果をクライアントに送る．クライアントはイニシエータデバイスと呼ばれ，計算機である．また，サーバは，ターゲットデバイスと呼ばれ，ディスクアレイやテープライブラリである．ターゲットデバイスは，1 個以上のロジカルユニットを保持する．ロジカルユニットは，イニシエータデバイスからの要求を処理するものであり，ディスクドライブやテープドライブである．

トランスポートは，イニシエータデバイスとターゲットデバイスを接続するための媒体である．SAN では，FC，Ethernet，IB 等のハードウェアが用いられる．Ethernet の場合，イニシエータデバイスとターゲットデバイスの両方に搭載されるハードウェアインタフェースとなる Network Interface Card（NIC）と，Ethernet ケーブル，スイッチが，トランスポートを構成する．イニシエータとターゲット間の情報のやりとりには，トランスポートに用いられるハードウェア特有のデータ形式が用いられる．このようなデータ形式やエラー処理等，トランスポート固有の機能は，トランスポートプロトコルとして定義されている．

2.1 ターゲットドライバ

ターゲットデバイスの主な役割は以下の 2 つである．

- トランスポートのハードウェアインタフェース制御とトランスポートプロトコルの処理．
- イニシエータから受け取った要求を処理し，適切な結果を送る．必要があれば，ロジカルユニットにアクセスする（ディスクドライブからデータを読み出す等）．

これらの機能を提供するソフトウェアは，一般にターゲットドライバと呼ばれる．トランスポートのハードウェアインタフェースは独自であるため，それぞれのハードウェアインタフェース専用のターゲットドライバが必要である．しかし，残りの役割の大半は，SAN プロトコルやトランスポートの種類に非依存であり，コードの重複を避けるため，各ターゲットドライバに共通な機能を提供するフレームワークが必要になる．

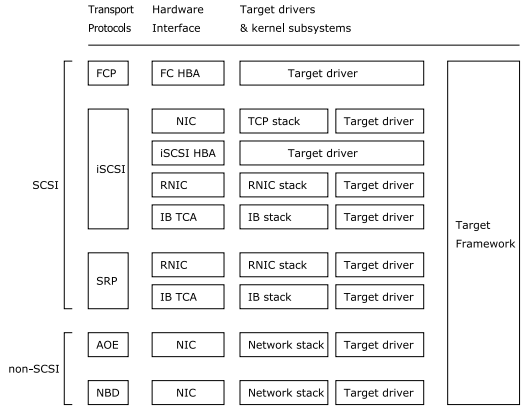


図 2 ストレージプロトコルとハードウェアインタフェース構成
Fig. 2 Protocols and hardware interfaces.

ターゲットドライバは，フレームワークにイニシエータからの要求の処理を依頼し，その結果を受け取り，イニシエータに送る．

図 2 は，ストレージプロトコル，ハードウェアインタフェース，ターゲットドライバ，フレームワークの関係を単純化したモデルである．図中には，SCSI プロトコルを使う SAN プロトコルとして，Fibre Channel Protocol（FCP），Ethernet 用の iSCSI [3]，IB 等 RDMA をサポートするネットワークで使用する SCSI RDMA Protocol（SRP）の 3 種類，SCSI を使わない SAN プロトコルとして，AOE と NBD を示した．

ターゲットドライバは，ハードウェアモデル，ソフトウェアモデルに分類される．

ハードウェアモデル ハードウェアインタフェースがトランスポートプロトコル処理の大部分を担当し，ターゲットドライバはハードウェアインタフェースとフレームワークをつなぐ役目を果たす．ハードウェアインタフェースはベンダごとに異なるため，ハードウェアインタフェース専用のターゲットドライバが必要となる．FC のハードウェアインタフェースである Host Bus Adapter（HBA）や iSCSI 専用ハードウェアインタフェース（iSCSI HBA）のターゲットドライバがハードウェアモデルに分類される．

ソフトウェアモデル NIC をハードウェアインタフェースとして利用する iSCSI の場合，NIC は iSCSI 専用の機能を提供しないため，ターゲットドライバは，トランスポートプロトコル処理のすべてを実行する必要がある．また，ターゲットドライバは，オペレーティングシステムの TCP/IP スタックを介して NIC にアクセスするため，1 つの iSCSI ターゲットドライバが，すべての NIC に

対応できる．Ethernet で RDMA をサポートするハードウェアインタフェースである RDMA NIC（RNIC），IB HCA（Host Channel Adapter）を利用した SRP でも，オペレーティングシステムの RDMA 用スタックを利用するため，SRP 用ターゲットドライバは 1 種類となる．ネットワークスタックを利用する AOE や NBD のターゲットドライバもソフトウェアモデルに分類される．

3. 実　　装

tgt の基礎となる設計方針は，その主要機能をユーザ空間で実現することである．カーネル空間に実装するのは，ハードウェアアダプタに直接アクセスするターゲットドライバとユーザ空間とカーネル空間でデータを転送するためのインタフェース機能だけである．tgt と異なり，既存のターゲットドライバのためのフレームワークや単体で動作するターゲットドライバは，すべての機能がカーネル空間に実装されている．

カーネル空間で実現されていた機能をユーザ空間に実装する手法は，過去にマイクロカーネル[4),5)] で試されてきた手法であり，カーネルクラッシュを避け信頼性を向上させる，コードの独立性を高めることで開発・保守を容易にするという利点が得られる．近年の Linux カーネル開発では，この設計手法が積極的に採り入れられている[6)~8)]．

バグ等の理由で，tgt のユーザ空間のプロセスがクラッシュした場合，イニシエータデバイスに要求の結果が送信されない．イニシエータデバイスの実装によってエラー処理は異なるが，一般的な実装では，一定時間結果が帰ってこないコマンドの実行中止を要求し，再試行を試みる．それでも，ターゲットデバイスが復旧しなければ，イニシエータデバイスは，ロジカルユニット，ターゲットデバイスのリセットによる復旧を試みる．イニシエータデバイスは，リセットによる復旧が失敗すると，ターゲットデバイスを利用不可能な状態として扱う．

現在，tgt が提供している復旧手段は，最も単純なユーザプロセスの再起動のみである．クラッシュによりイニシエータデバイスが接続していたという情報や，実行中だったコマンドに関する情報は失われるため，イニシエータデバイスでの復旧処理も必要である．具体的には，イニシエータデバイスは，ターゲットデバイスに再接続し，その状態を利用可能に変更する．イニシエータデバイスのターゲットデバイスへの再接続は，定期的に再接続を試みる，ユーザの明示的な再接続操作等，イニシエータデバイスの設定によって異なる．

表 1 カーネル空間ソフトウェアコンポーネント
Table 1 tgt software components in kernel space.

名称	機能
tgt core	ストレージプロトコル非依存，ハードウェアアダプタ非依存の共通機能を提供する．
プロトコルモジュール	ストレージプロトコル依存機能を提供する．ストレージプロトコルごとに用意される．
TTLD	ハードウェアアダプタ依存機能を提供する．

表 2 ユーザ空間ソフトウェアコンポーネント
Table 2 tgt software components in user space.

名称	機能
tgtd	コマンド処理以外のストレージプロトコル処理，システム全体の管理機能を提供するデーモンプログラム．1 プロセスだけ起動する．
cmdd	ストレージプロトコルのコマンド処理用のデーモンプログラム．ターゲットデバイスごとに 1 プロセスが起動する．
tgtadm	統一された管理インタフェースをユーザに提供するユーティリティ．tgtd に接続して，ユーザからの管理要求を伝える．
プロトコルライブラリ	ストレージプロトコル依存機能を提供するダイナミックリンクライブラリ．tgtd と cmdd がロードする．

tgt は，同時に複数のターゲットデバイスのインスタンスを動作させることができ，1 つのターゲットデバイスは，複数のロジカルユニットを管理することができる．ロジカルユニットの種類はプロトコルによって異なるが，SCSI プロトコルの場合，ディスクドライブ，テープドライブ，光学ドライブ等の種類がある．本稿では，ディスクドライブを対象とする．

3.1 ソフトウェアコンポーネント構成

表 1，**表 2** に，それぞれ，カーネル空間，ユーザ空間の tgt のソフトウェアコンポーネントの機能をまとめた．また，それらのソフトウェアコンポーネントの関係を**図 3** に示した．

tgt のカーネルコンポーネントは，ターゲットドライバとユーザ空間を接続し，情報を運ぶ単純なパイプのように働き，イニシエータから届いたコマンドはユーザ空間で処理される．

カーネル空間のコンポーネントは，カーネルモジュールとして実現されており，必要なモジュールのみが動的にロードされる．

ユーザ空間のコンポーネントは，2 種類のデーモンプログラムとプロトコルライブラリから構成される．プロトコルライブラリはダイナミックリンクライブラリとして実装されており，必要なライブラリだけがロードされる．

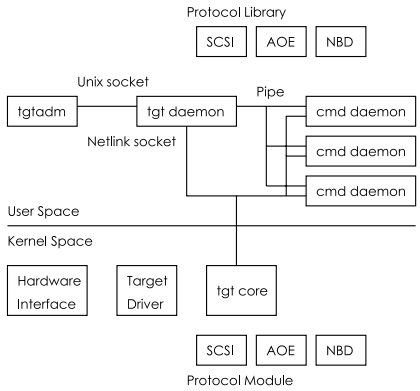


図 3 ソフトウェアコンポーネント構成
Fig. 3 tgt software components.

3.2 ターゲットドライバ

ターゲットドライバは，ハードウェアアダプタに直接アクセスする必要があるため（ハードウェア割込み等），カーネル空間に実装されている．一般的なハードウェアモデルのターゲットドライバの動作は以下のとおりである．

(1) 初期化時に，ターゲットドライバは，tgt core にストレージプロトコルの種類を宣言する．
(2) 新たにイニシエータデバイスにサービスを開始したときには，tgt core に通知する．
(3) イニシエータデバイスからコマンドを受け取り，tgt core に渡す．
(4) tgt core からバッファの準備完了通知を受け取る．
(5a) ターゲットデバイスがイニシエータデバイスにデータを送信するコマンドの場合は，ターゲットドライバは，tgt core から，レスポンスが保存されているバッファのアドレスを受け取り，ネットワークに送信するようにハードウェアインタフェースに命令する．
(5b) イニシエータデバイスからターゲットデバイスにデータを送信するコマンドの場合は，ターゲットドライバは，tgt core から，イニシエータデバイスからのデータを保存するバッファのアドレスを受け取り，ハードウェアインタフェースにバッファの準備ができたことを通知する．ハードウェアインタフェースはイニシエータデバイスにデータを送るように要求し，届いたデータを指定されたバッファに転送する．
(6) ハードウェアインタフェースがバッファからのデータ処理を終えたら，ターゲットドライバは，tgt core にコマンドの完了を通知する．

ソフトウェアモデルのターゲットドライバの場合，ハードウェアインタフェースに命令する代わりに，オペレーティングシステムの他のサブシステム（TCP スタック等）に命令する．

3.3 tgt core

tgt core は，ターゲットドライバからコマンドを受取りユーザ空間のデーモン（tgtd と cmdd）に転送する，デーモンからコマンドを受け取りターゲットドライバに転送する，という 2 つの役割を持つ．

ターゲットドライバからのコマンド受け取り操作では，tgt core は，以下の情報を受け取る．

- ターゲットデバイスの識別番号
- コマンドが含まれたバッファ（コマンドバッファ）
- コマンドバッファの長さ
- ロジカルユニットを特定するためのデータが含まれたバッファ（デバイスバッファ）
- デバイスバッファの長さ
- イニシエータ・ターゲットデバイス間でやりとりするデータの長さ
- データの送信方向（イニシエータデバイスからターゲットデバイス，または，ターゲットデバイスからイニシエータデバイス）

netlink と呼ばれる機能を使って，tgt core とデーモンは接続されている．netlink は，ユーザ空間のプログラムが，ソケット API を使って，カーネル空間とデータをやりとりするための機構である．

tgt core は，プロトコルモジュールの機能を呼び出し，上記の情報をプロトコルに応じた特定のデータフォーマットのパケットにして，デーモンに転送する．

tgt は，高速なプロトコル処理を実現するために，2 種類のデーモンプログラムを使っている．SCSI プロトコル場合，tgt core は，通常の SCSI コマンドを cmdd に転送する．通常の SCSI コマンドは固定長のため，cmdd は効率良くソケットからコマンドを取り出すことができる．SCSI コマンド以外の制御用コマンド（Task Management Functions 等）は tgtd に転送される．制御用コマンドは可変長のため，tgtd は，まず，ソケットからヘッダだけを読み出し，データ長を調べてから，コマンドのすべてのデータを読み出す．

3.4 プロトコルモジュール

カーネル空間に実装する必要があるプロトコル依存部分は非常に小さく，SCSI プロトコルモジュールの場合，前述の特定のフォーマットのパケットを組み立てる機能のみを提供し，コメントを除いて 80 行である．

3.5 tgtd

tgtd は，システム全体で 1 つだけ起動し，2 つの

機能を持つ．1 つ目の機能は，ターゲットデバイスやロジカルユニットのインスタンス作成・削除等の tgt 自体の管理機能である．後述するように，ユーザは，tgtadm を使って管理命令を tgtd に伝えることができる．tgtd は，tgtadm からの命令をルーティングする役割を持ち，必要に応じて対応するプロトコルライブラリを使い，命令をプロトコルに応じた特定のデータフォーマットのパケットにして，netlink ソケット経由で tgt core や，パイプ経由で cmdd に送信し，命令に対する結果を tgtadm に返す．

2 つ目の機能は，ターゲットドライバから tgt core 経由で届いた制御用コマンドを適切な cmdd に届け，cmdd から受け取った応答をターゲットドライバに返信することである．

3.6 cmdd

cmdd は，ターゲットドライバごとに，tgtd の子プロセスとして起動される．起動時に，ターゲットドライバのストレージプロトコルに応じたプロトコルライブラリをロードする．

cmdd の役割は，ストレージプロトコルのコマンド処理である．cmdd は，tgt core との間に確立された netlink ソケットから読み出すことで，コマンドを含むパケットを 1 つ取り出す．プロトコルライブラリを使って，コマンドを処理し，レスポンスを生成する．最後に，cmdd は，コマンドのレスポンスを，netlink ソケットに書き込むことで，tgt core に伝える．

ユーザ空間とカーネル空間でソケットインタフェースを通じてデータを交換する際に生じる問題は，メモリコピーによる性能低下である．特に，大量のデータをコピーする必要がある，ロジカルユニットからの読み書きを実行する I/O コマンドの処理が大きな問題となる．

tgt では，メモリマップ I/O と呼ばれる手法を使い，メモリコピーを回避している．cmdd は，tgt core にバッファの内容をソケットを通じて送信するのではなく，バッファをユーザ空間にマップし，tgt core にそのアドレスを送信する．

I/O コマンドに関しては，cmdd が mmap システムコールを使い，ロジカルユニットの指定された位置のページキャッシュをユーザ空間に貼り付け，mmap システムコールの返り値であるアドレスと貼り付けた長さを tgt core に送信する．その後，cmdd は，tgt core からコマンドの終了通知を受け取ると，munmap システムコールを使って，ページキャッシュをアンマップする．

I/O コマンド以外のロジカルユニットからの読み書きが発生しないコマンドの場合，cmdd は，malloc で確保したユーザ空間のバッファにレスポンスを作成し，そのアドレスと長さを tgt core に渡す．コマンドの終了通知を受け取ると，cmdd は確保したメモリを解放する．

上記のいずれの場合も，tgt core は，cmdd から受け取ったアドレスとそのプロセス構造体から，適切なバッファを判断し，そのアドレスをターゲットドライバに通知することができる．

ユーザが，ターゲットデバイスにロジカルユニットを追加した際，cmdd は，ロジカルユニット全体の mmap を試みる．成功した場合，cmdd は，コマンドごとの mmap，munmap システムコールを省略することで，性能を向上させる．tgt では，この最適化を，mmap キャッシュと呼ぶ．mmap キャッシュによる最適化を使うためには，ロジカルユニットの大きさがプロセス空間に収まる必要がある．プロセス空間が広大な，64 bit アーキテクチャでは，多くの状況で利用可能である．

tgt では，通常のファイルやブロックデバイスファイルを，ロジカルユニット（ディスクドライブ）として扱うことができる．また，Linux カーネルが提供する，容量の動的な増減機能や冗長性機能等を持つ仮想ディスクドライブ[9] もロジカルユニットとして用いることができるため，商用ターゲットデバイス同様の柔軟な管理が可能である．

3.7 プロトコルライブラリ

プロトコルライブラリは，イニシエータから届いたコマンドの実行，レスポンスの生成，また，制御用の要求を処理する．

SCSI プロトコルの場合，コマンドの解析は，コマンドが含まれたバッファ（SCSI Control Block）からのコマンド種類と内容の判断，ロジカルボリューム情報が含まれたバッファからのデバイス番号（Logical Unit Number）の判断，から成り立っている．

3.8 tgtadm

tgt は，ストレージプロトコルに非依存な管理インタフェースを提供する．システムの管理用のデフォルトのユーティリティプログラムとして，tgtadm が提供される．tgtadm は，Unix ドメインソケットを使って tgtd に接続し，管理用の命令を伝え，tgtd から受け取った結果を出力する．管理者は，tgtadm を使わず，Unix ドメインソケットを使って，tgtd を操作する独自の管理用プログラムを実装することもできる．

4. 性能評価

ユーザ空間でのストレージプロトコル処理による性

表 3 サーバ構成
Table 3 Experimental infrastructure.

CPU	Intel Xeon 2.8 GHz × 2
Memory	4 GB
NIC	Intel Pro/1000
OS	Linux kernel 2.6.15（64 bit カーネル）

能への影響を調べるため，tgt と NIC 用の iSCSI ターゲットドライバ（istgt）の組合せとカーネル空間で動作する NIC 用の iSCSI ターゲットドライバ，iSCSI Enterprise Target（IET）[10] の性能を比較した．IET は，スタンドアローン（単体で動作する）の NIC 用の iSCSI ターゲットドライバで，エントリクラスの商用ストレージデバイスと同程度の性能が達成できる．

istgt は，IET のコードを流用しており，iSCSI プロトコル処理等，tgt が提供しない機能に関しては，ほぼ同じコードになっている．

4.1 実験環境

2 台の計算機を Extreme 社の Summit 7i Gigabit Ethernet スイッチを介して接続し，それぞれを iSCSI イニシエータ，ターゲットとして動作させた．実験で使用したハードウェアとソフトウェアについて**表 3** にまとめた．

ターゲットデバイスは，LSI Logic 社の 53C1030 チップを搭載した Ultra320 SCSI ホストバスアダプタ経由で接続されている，Seagate 社の 15000 回転の SCSI ドライブ（Cheetah ST336753LC）を，ロジカルユニットとして利用した．

ターゲットデバイスのディスクキャッシュのポリシは，write back を選択した．イニシエータデバイスからの WRITE コマンドが更新したデータを，すぐにディスクドライブには書かず，ページキャッシュに保存した時点で，イニシエータデバイスにコマンドの完了を通知する．

iSCSI イニシエータドライバは，標準 Linux カーネルに含まれている open-iscsi [8] を利用した．iSCSI プロトコルのパラメータは，RFC に定められたデフォルト値を利用した．

結果は 5 回の測定の平均値である．

4.2 マイクロベンチマーク

4.2.1 プロトコル処理時間

図 4 に，tgt と IET が，1 個の SCSI コマンドの処理にかかる時間の測定結果を示した．

I/O サイズが 4 KB 以下の READ 命令に関しては，IET のプロトコル処理時間が tgt よりも数十マイクロ秒短いが，大きな差は生じていない．それよりも大きい I/O サイズで，tgt のプロトコル処理時間が IET よりもわずかながら短い理由は，tgt がメモリコピーをせずに SCSI コマンド処理するのに対して，IET のプロトコル処理ではメモリコピーが発生するためである．I/O サイズが大きくなると，メモリコピーに必要な時間は長くなり，プロトコル処理に占める割合も大きくなる．

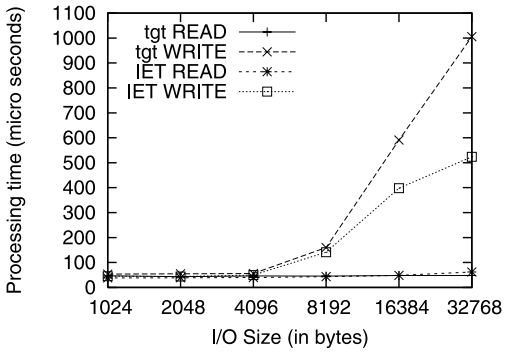


図 4 プロトコル処理時間
Fig. 4 Protocol processing time.

I/O サイズが 8 KB 以下の WRITE 命令に関しては，READ 命令同様に，IET のプロトコル処理時間が tgt よりも数十マイクロ秒短いが，大きな差は生じていない．8 KB で，tgt と IET の処理時間が増加する理由は，ページサイズの 4 KB を超え，複数のページフレームの I/O が必要になるからである．

I/O サイズが 16 KB 以上で，tgt と IET のプロトコル処理時間がさらに増加する理由は，今回の iSCSI パラメータの設定条件では，iSCSI のコマンド処理方法が変化し，プロトコル処理時間にネットワーク遅延が含まれるからである．I/O サイズが 8 KB 以下の場合，イニシエータデバイスは WRITE コマンドと連続して，更新データをターゲットに送信できるが，16 KB 以上では，イニシエータデバイスは WRITE コマンドと 8 KB の更新データを送信し，ターゲットからの R2T コマンド☆を待ってから，残りの更新データを送信する．

4.2.2 シーケンシャル I/O 性能

disktest ベンチマーク [11] をイニシエータデバイスで動作させて，ターゲットデバイスの基本的な性能上の特徴を調査した．disktest ベンチマークは，Direct I/O 機能を利用しており，イニシエータデバイスのページキャッシュの影響を受けない．

I/O サイズを 2 KB〜32 KB に設定し，それぞれ，

☆ ターゲットは更新データを保存するバッファを準備してから，Ready To Transfer（R2T）コマンドをイニシエータに送信する．

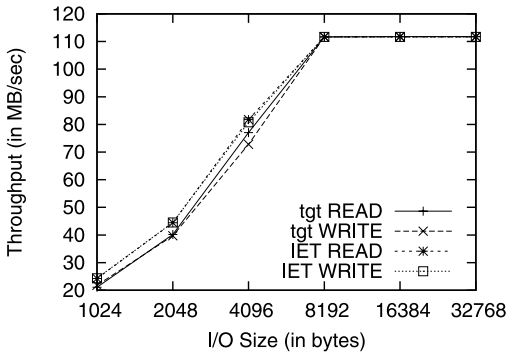


図 5 シーケンシャル I/O 性能
Fig. 5 Sequential results.

30 秒間のシーケンシャルなディスク IO 命令を発生させた．

図 5 に測定結果を示す．I/O サイズが 4 KB 以下の場合，tgt の性能は，IET の性能の 87.1%〜94.2%であった．一方，I/O サイズが 8 KB 以上の場合，IET，tgt の性能に差はなく，いずれも，Ethernet の帯域性能の限界近くの性能を達成できることが分かった．

I/O サイズが小さい場合，ターゲットデバイスでの処理時間が性能に与える影響が大きい．tgt と IET の SCSI コマンド処理時間の差が，シーケンシャルアクセス性能差の理由である．

一方，I/O サイズが大きくなると，データ転送時間が長くなるため，ターゲットデバイスでのコマンド処理時間の性能への影響が小さくなる．そのため，I/O サイズが大きい場合，IET のプロトコル処理時間は tgt よりもかなり高速であるが，それぞれのシーケンシャル性能には差が生じていない．

4.3 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，イニシエータデバイスで，Postmark ベンチマーク[12] と dbench[13] を実行した．

イニシエータデバイスはターゲットデバイスが提供するロジカルユニット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する．

Postmark は，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．ファイルサイズを 512 B から 16 KB，ファイル数を 20,000 個，トランザクション回数を 50,000 回に設定している．dbench は，ファイルサーバの負荷を模擬する．クライアント数を 32，実行時間を 60 秒に設定した．

表 4 マクロベンチマークの結果
Table 4 Macrobenchmark results.

	postmark	dbench
	Throughput (transactions per second)	Throughput (MB/s)
tgt	1214.2	715.1
IET	1250.0	719.7

表 4 に Postmark と dbench の結果を示す．ブロックサイズが 4 KB のため，イニシエータデバイスから届く SCSI コマンドの I/O サイズは，つねに 4 KB 以上になる．その結果，シーケンシャル I/O の測定結果から予想できるように，tgt と IET の差は非常に小さく，それぞれのベンチマークで，tgt は，IET の 97.1%，99.4%の性能を達成できている．

5. 関連研究

ユーザ空間にカーネル空間の機能を追い出す手法は，主に，マイクロカーネル研究で，その有用性が示されてきた．マイクロカーネルは，オペレーティングシステムの動作に必須の機能をユーザ空間に実装している．一方，tgt の採用している手法は，モノリシックカーネルで，オペレーティングシステムの動作には影響しない機能をユーザ空間に実装するもので，マイクロカーネルの手法とは大きく異なる．

マイクロカーネルの場合，カーネル機能を実現しているユーザプロセスがクラッシュした場合，多くの場合，計算機の再起動が唯一の復旧手段である[14]．一方，tgt のユーザプロセスがクラッシュした場合でも，オペレーティングシステムはその影響を受けることなく動作し続けることができ，前述のように，tgt のプロセスを再起動することで，イニシエータデバイスはターゲットデバイスに再接続が可能となる．

tgt のプロセスの再起動後，カーネル空間からクラッシュ前の状態を得るインタフェースを実装することで，クラッシュによるイニシエータへの影響を小さくすることができる可能性がある．たとえば，Linux の SCSI イニシエータデバイスの，コマンドのデフォルトタイムアウト時間は 30 秒であり，その間にクラッシュ前の状態が回復できれば，イニシエータデバイスにクラッシュの影響が発生しない．

ソフトウェアバグによってプロセスがクラッシュした場合，たとえば，主要な機能にバグがあれば，じきに，プロセスが再びクラッシュする可能性が高い．このようなバグは事前のテストで発見できる可能性が高いため，単純なプロセスの再起動は有効な復旧手段ではあるが，より広範囲のバグに対応するための様々な

手法[15]が提案されており，これらを適用することで，さらなる信頼の向上が期待される．

マイクロカーネルと異なり，カーネルの動作とユーザプロセスが依存していないため，プロセスの実装が単純になることも利点である．たとえば，マイクロカーネルの pager プロセスには，メモリの循環依存関係を避けるために実装が複雑になるという問題がある[16]．

ソフトウェアの信頼性を向上させる他のアプローチとしては，カーネル内部のソフトウェア障害からの復帰を目指した機能をカーネルに追加する手法[14]，カーネル内での安全なコードの実行を目標とした，型安全なプログラミング言語でカーネルコードを実装する手法[17]やコンパイラとカーネルへの拡張を利用した実行時の動的チェックによる手法[18]等が提案されている．tgt は，これらの手法と異なり，既存のカーネルの機能だけを利用しており，カーネル内部の障害には対応できないが，カーネルで動作するコードの大部分を安全に実行（復旧）可能なユーザ空間に追い出すことで，信頼性の向上を目指した．tgt のカーネル内コードにこれらの手法を適用することで，さらに信頼性を向上させることが可能である．

データの代わりに，そのデータのメモリアドレスを使うデータ転送の高速化手法は，Linux カーネルの SCSI generic driver[19]や Direct I/O や，様々なマイクロカーネルの IPC で広く使われている．また，モノリシックカーネルの仮想記憶に変更を加えること，カーネル・ユーザ間の高速データ転送で実現した IO-Lite[20]等の例もある．

SCST（Generic SCSI Target Middle Level）[21]は，SCSI ターゲットドライバのためのフレームワークを実装した唯一の試みである．tgt との最大の違いは，SCST のすべての機能はカーネル空間に実装されていることである．

ibmvscsis[22]は，IBM 社 iSeries の仮想化環境用のカーネル内で動作するスタンドアローンの SRP ターゲットドライバである．iSeries の仮想化環境では，VIO サーバと呼ばれる特別な仮想マシンがターゲットデバイスになり，残りの仮想マシンがイニシエータデバイスとなることで，ストレージ仮想化を実現する．仮想化環境では，RDMA 同様，仮想マシン間でメモリに直接アクセスし，データ転送が可能である．tgt を使うように ibmvscsis を書き換えたところ，2,000 行以上のコードを削減することができた．

本稿では，Linux に実装した tgt について述べたが，FreeBSD 等，他のモノリシックなカーネルのオペレーティングシステムでも，容易に実装できる．tgt のカーネル部分のコードは，Unix カーネルの標準的な機能を使い，コメントを含めて 2,000 行以下であるため，容易に移植可能である．ユーザ空間のコードは，カーネルとユーザ間のデータ通信インタフェースである netlink 部分のみが Linux カーネルに依存しており，そのコード量は 300 行以下である．netlink は，カーネルとユーザ間のデータ通信を提供する機能，たとえば，Unix カーネルの標準的な機能である ioctl 等で置き換えることができる．

6. ま と め

本稿では，ストレージプロトコル処理等，主要な機能をユーザ空間に実装し，カーネル内の実装を最小限にすることで信頼性を向上させた，SAN ターゲットドライバ用フレームワーク，tgt を Linux に実現し，その評価を行った．

ターゲットドライバは，tgt の機能を利用することで，その実装を簡素化できるとともに，信頼性を向上させることができる．tgt はストレージプロトコルの種類に非依存なため，FCP，iSCSI，SRP 等の各種 SCSI プロトコルに加え，AOE，NBD 等の非 SCSI プロトコル用の各種 SAN プロトコルのターゲットドライバが tgt を利用することができる．

tgt は，メモリマップ I/O を用いたメモリコピー回避等の手法を使うことで，ユーザ空間の利用にともなう性能低下をおさえた．商用環境を模擬した環境で，その性能を評価したところ，カーネル空間に実装されたターゲットドライバと比較して，性能の低下は 3%未満であった．

`http://stgt.berlios.de/` から，tgt のソースコードは入手可能である．カーネル部分のコードは，開発版の Linux カーネルにも含まれている．

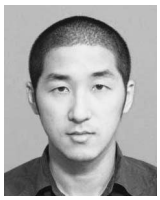
謝辞 共同開発者である Mike Christie，設計に関する有益な助言をくださった Christoph Hellwig，James Bottomley に感謝の意を表する．

参 考 文 献

1) Coile, B. and Hopkins, S.: The ATA over Ethernet Protocol (2004). http://www.coraid.com/documents/AoEr8.txt
2) Machek, P.: Network Block Device. http://nbd.sourceforge.net/
3) Satran, J., Meth, K., Sapuntzakis, C., Chadalapaka, M. and Zeidner, E.: Internet Small Computer Systems Interface (iSCSI), RFC 3720 (2004).

4) Accetta, M., Baron, R., Bolosky, W., Golub, D., Rashid, R., Tevanian, A. and Young, M.: Mach: A new kernel foundation for UNIX development, *Summer 1986 USENIX Conference*, pp.93–112 (1986).
5) Liedtke, J.: On μ-kernel construction, *the 15th ACM Symposium on Operating Systems Principles*, pp.237–250 (1995).
6) Szeredi, M.: Filesystem in Userspace (2001). http://fuse.sourceforge.net/
7) Goggin, E., Kergon, A., Varoqui, C. and Olien, D.: Linux Multipathing, *Ottawa Linux Symposium*, pp.147–167 (2005).
8) Yusupov, D. and Aizman, A.: Open-iSCSI. http://www.open-iscsi.org
9) Teigland, D. and Mauelshagen, H.: Volume Managers in Linux, *the USENIX Annual Technical Conference*, Boston, MA, pp.185–198 (2001).
10) 藤田智成，小河原成哲：iSCSI ターゲットソフトウェアの解析，情報処理学会論文誌：コンピューティングシステム，Vol.46, No.SIG 3 (ACS 8), pp.38–50 (2005).
11) SGI and IBM: The Linux Test Project test suite. http://ltp.sourceforge.net/
12) Katcher, J.: PostMark: A New File System Benchmark, Technical Report TR3022, Network Appliance (1997).
13) Tridgell, A.: dbench benchmark (2001). http://samba.org/ftp/tridge/dbench/
14) Swift, M.M., Bershad, B.N., H. M.L.: Improving the Reliability of Commodity Operating Systems, *the 19th ACM Symposium on Operating Systems Principles*, pp.207–222 (2003).
15) Qin, F., Tucek, J. and Zhou, Y.: Treating Bugs as Allergies: A Safe Method for Surviving Software Failures, *10th Workshop on Hot Topics in Operating Systems* (2005).
16) Hand, S., Warfield, A., Fraser, K., Kotsovinos, E. and Magenheimer, D.: Are Virtual Machine Monitors Microkernels Done Right?, *10th Workshop on Hot Topics in Operating Systems* (2005).
17) Bershad, B.N., Savage, S., Pardyak, P., Sirer, E.G., Fiuczynski, M.E., Becker, D. and Susan Eggers, C.C.: Extensibility, safety and performance in the SPIN operating system, *the 15th ACM Symposium on Operating Systems Principles*, pp.267–284 (1995).
18) Purohit, A., Wright, C.P., Spadavecchia, J. and Zadok, E.: Cosy: Develop in User-Land, Run in Kernel-Mode, *9th Workshop on Hot Topics in Operating Systems*, pp.109–114 (2003).
19) Douglas Gilbert: The Linux SCSI Generic (sg) Driver. http://sg.torque.net/sg/
20) Pai, V.S., Druschel, P. and Zwaenepoel, W.: IO-Lite: A Unified I/O Buffering and Caching System, *ACM Trans. Comput. Syst.*, Vol.18, No.1, pp.37–66 (2000).
21) Bolkhovitin, V.: Generic SCSI Target Middle Level for Linux (2003). http://scst.sourceforge.net/
22) Boutcher, D. and Engebretsen, D.: Linux Virtualization on IBM POWER5 Systems, *Ottawa Linux Symposium*, pp.113–120 (2004).

(平成 18 年 1 月 27 日受付)
(平成 18 年 6 月 21 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．