IP-SAN トレースシステムを用いたストレージアクセス解析

山口　実靖† 　小口　正人†† 　喜連川　優†

† 東京大学生産技術研究所　〒 153-8505 目黒区駒場 4-6-1
†† お茶の水女大学理学部情報科学科 〒 112-8610 東京都文京区大塚 2-1-1
E-mail: †{sane,kitsure}@tkl.iis.u-tokyo.ac.jp, ††oguchi@computer.org

あらまし　本稿では iSCSI を用いた IP-SAN のアクセストレースシステムの提案と, それを用いた IP-SAN システムの振る舞いの解析を行う. FC-SAN の欠点を補う SAN として IP を用いる IP-SAN や iSCSI が期待を集めるようになっているが, IP-SAN は性能が FC-SAN より劣るという問題点も指摘されている. IP-SAN は多段のプロトコルで構成され, かつ サーバ計算機とストレージ機器が協調して動作する分散システムとなっている. よって, これらの統合的な解析の実現が重要である. 本稿では, 我々の実装した統合的なトレースシステムについて説明を行う. そしてそれを用いてファイルシステムやネットワークなど IP-SAN システム (Linux OS と ニューハンプシャー大学の iSCSI 実装を使用) の各層の振る舞いの解析を行い, 性能の解説を行う.

キーワード　ネットワークストレージ IP ストレージ IP-SAN

Storage Access Analysis using IP-SAN Trace System

Saneyasu YAMAGUCHI†, Masato OGUCHI††, and Masaru KITSUREGAWA†

† Institute of Industrial Science, University of Tokyo　4-6-1 KOMABA MEGURO-KU, TOKYO 153-8505, JAPAN
†† Department of Information Sciences Faculty of Science Ochanomizu University, 2-1-1 Otsuka Bunkyo-ku, Tokyo, 112-8610 Japan
E-mail: †{sane,kitsure}@tkl.iis.u-tokyo.ac.jp, ††oguchi@computer.org

Abstract　In this paper, we propose an integrated IP-SAN trace system using iSCSI protocol. Protocol stack of IP-SAN system is composed of many protocols, and server computers and storage appliances works cooperatively in IP-SAN system. Thus understanding its behavior is difficult and constructing a tool which can do an integrated analysis including all layers in both server computers and storage appliances. We proposed an integrated IP-SAN trace system and show detailed explanation of performances.

Key words　Network Storage, IP Storage, IP-SAN

1. はじめに

近年, 計算機システムの性能向上やストレージの大容量化, 高性能化に伴い, 計算機システムが扱うデータの量が増大している. その結果ストレージ管理費用が増大し, この問題が計算機システムの大きな問題の一つとなっている [1], [2]. この問題に対する解決策の一つとして, SAN (Storage Area Network) の導入によるストレージ集約が提案された. 計算機群が使用するストレージを一箇所に集約して配置し各計算機がネットワーク (SAN) 経由でこれを使用する運用手法を取ることにより, ストレージ管理費用は大幅に削減される. この効果は高く評価されておりすでに多くの企業で SAN が導入されている. しかし, FC を用いる現世代の SAN (“FC-SAN”) は, (1)FC 管理技術者が少ない, (2)FC は接続距離に限界がある, (3)FC 導入費用は高い, (4)FC の相互接続性は必ずしも高くない, などの問題も指摘されており, Ethernet と TCP/IP を用いて構築する IP-SAN が 次世代 SAN として期待を集めている. IP-SAN は, IP 技術者が多い, 接続距離に限界がない, 導入費用が低い, 相互接続性が高いなどの利点が期待されている. 逆に欠点としては, 性能が FC-SAN より低い, サーバ計算機の CPU 使用率が高くなる, などが指摘されている [3]〜[5].

IP-SAN 用のデータ転送プロトコルとしては iSCSI [6], [7] が 2003 年 2 月に IETF [6]〜[8] に承認され, これが標準的なデータ転送プロトコルとなっている. そこで本稿では iSCSI ストレージアクセスのトレースシステムについて述べる. iSCSI では, SCSI プロトコルを TCP プロトコルの中にカプセル化し

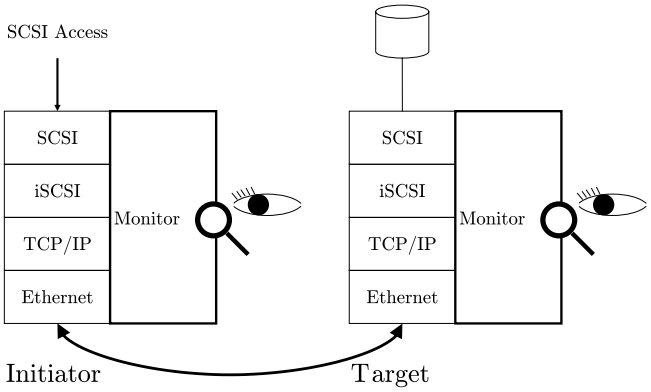

図 1　iSCSI プロトコルスタックと解析システム

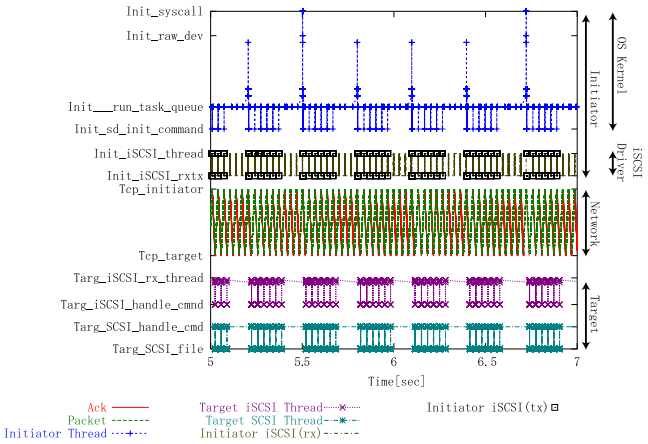

図 2　iSCSI アクセスのトレース

IP ネットワーク上で転送するブロックレベルのプロトコルである. 多くの場合, 物理層, トランスポート層には Ethernet が用いられ, 代表的なプロトコルスタックは図 1 の様に SCSI over iSCSI over TCP/IP over Ethernet となる. さらに, ほとんどの場合 SCSI 層の上位にはさらにファイルシステムやブロックデバイスが配置される. このように, IP-SAN システムはプロトコルスッタックが多段であり, その振る舞いを把握することが困難である. そこで我々は, 文献 [9] においてアプリケーションにより発行された I/O 要求が各層を通過しディスクアクセスに到るまでを追跡が可能な “トレースシステム” を提案した. そして, 試作システム (raw デバイスを使用しておりファイルシステムの振る舞いの把握ができない) を実装し, 応用例を紹介しその有効性を示した. 本稿では, より高度なトレースや解析の実現として, (I/O 要求を発行する) アプリケーションと IP-SAN システムの統合的なトレースを提案する. また, 文献 [9] において実現されていなかったファイルシステムを含めたトレースの試作実装についての紹介を行う.

本稿は以下のように構成される. 第 2. 章において提案する “トレースシステム” の紹介を行う. そして, 第 3. 章において提案システムを用いて raw デバイスを用いた IP-SAN システムをトレースした例を紹介する. 次に, 第 4. 章において, より現実的な応用の解析の例としてファイルシステム上で動作するアプリケーションおよびファイルシステムを含む IP-SAN システムの解析を提案し, その試作実装の動作の紹介を行う. 第 5. 章で関連する研究の紹介を行い, 最後に第 6. 章において本稿をまとめる.

2.　統合トレースシステム

本章では提案する “IP-SAN アクセストレースシステム” について説明を行う. iSCSI ストレージアクセスは個別に動作するサーバ計算機とストレージ機器が協調して実行されるため, この双方の振る舞いを統合的に解析することが重要であると考えられる. また iSCSI ストレージアクセスは多段のプロトコルで構成されるが, 全層を経由して行われるため全層が性能劣化原因となる可能性があり, 性能向上について考察するにはこれら全層を網羅的に観察する必要がある. そこで図 1 の様に iSCSI プロトコルスタック全層の振る舞いを観察できるモニタシステムを構築し [10], サーバ計算機とストレージ機器において個別に記録された iSCSI ストレージアクセスのトレース記録を統合的に解析することを可能とするトレースシステムを構築した. 実装には, オープンソースである Linux OS(ver. 2.4.18) とニューハンプシャー大学 InterOperability Lab [11] が提供する iSCSI reference implementation (ver. 1.5.02) [12] を用い (以下この iSCSI 実装を “UNH” と呼ぶ), これらのソースコードにモニタ用コードを適用することにより解析システムを実現させた.

図 2 に, 当システムを用いて iSCSI シーケンシャルアクセスのトレース結果を可視化した例を示す. 同図の縦軸は iSCSI アクセスのプロトコルスタックの遷移を表している. すなわち, 上から順に, (1) アプリケーションによるシステムコールの発行, (2)raw デバイス層, (3)SCSI 層, (4)iSCSI 層, (5)TCP/IP 層, (6)Ethernet によるパケットの転送, (7)TCP/IP 層, (8)iSCSI 層, (9)SCSI 層, (10)HDD デバイスアクセス, を表している. (1)〜(5) が, サーバ計算機内における処理であり, (7)〜(10) がストレージ機器における処理である. 本トレース例では, ファイルシステムを用いずに raw デバイスを用いた. また使用した iSCSI ターゲットは “ファイルモード” で動作させたため最下位層の HDD デバイスアクセスは実際はファイルアクセスのトレースとなっている. 横軸が時間の経過を表しており, iSCSI ストレージアクセスの各層における各処理の消費時間等を視覚的に確認することが可能となる. また, 大きなブロックサイズのシステムコールが各層で細分化されている様子や, 待ち状態にある処理の把握なども可能となる. 同図の例では 2MB のシステムコールが発行されており, これを raw デバイスが 512KB ごとの 4 要求に分割し, 4 要求が完了した時点で上位層にシステムコールの完了を通知していることや, raw デバイスから発行された 512KB の要求が 32KB の SCSI 命令に分割されていること, 細分化された要求を用いてネットワークに処理要求が送出されている様などを観察することが可能である.

3.　トレースシステムの適用例

本章では提案トレースシステムを実際に高遅延環境下における並列 iSCSI ショートブロックアクセスに対し適用した例を紹

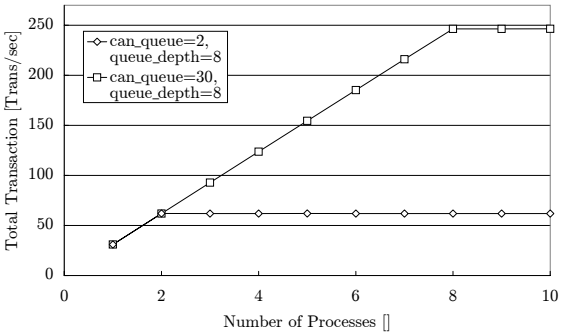

図 3　実験結果 A：並列 I/O の合計性能

介し, その有効性を示す.

3.1　並列アクセス実験

iSCSI イニシエータ (サーバ計算機) と iSCSI ターゲット (ストレージ機器) を Gigabit Ethernet で接続し IP-SAN 環境を構築し性能測定実験を行った. イニシエータとターゲットの間には人工的な遅延装置として FreeBSD Dummynet [13] を配置し擬似的な高遅延環境を実現した. iSCSI 実装としては, UNH 実装を用いた. iSCSI プロトコルの振る舞いの影響とストレージデバイスの振る舞いを明確に分離するために, ターゲットはファイルモードで動作させ, ファイル内容がメインメモリにキャッシュされている状態で計測を行った. イニシエータ, ターゲット PC の仕様は下記の通りである. CPU は Pentium4 2.8GHz, メインメモリは 1GB, OS は Linux 2.4.18-3, NIC は Intel PRO/1000 XT Server Adapter(Gigabit Ethernet Card). 遅延装置の PC の仕様は下記の通りである. CPU は Pentium4 1.5GHz, メインメモリは 128MB, OS は FreeBSD 4.5-RELEASE, NIC は Intel PRO/1000 XT Server Adapter.

本実験環境において以下ような並列ショートブロックアクセス実験を行った. サーバ計算機からストレージ機器に対して iSCSI 接続を確立し, その raw デバイスに対してシーケンシャルにショートブロックアクセス (raw デバイスに対してシステムコール "`read()`") を行うベンチマークプログラムを作成し, 同プログラムを複数プロセス同時に起動し全プロセスの合計性能を測定した. ブロックサイズは 512 バイトとし各プロセス 2048 回システムコールを発行した. 片道遅延時間は 16m 秒とした.

3.2　実 験 結 果

解析実行前の初期状態において上記の実験を行い図 3 の "can_queue=2, queue_depth=8" の結果を得た. 横軸が並列に動作させたベンチマークプロセスの数である. 縦軸は合計性能を表し, 単位時間における全プロセスの合計トランザクション数である (トランザクション数は "512 バイトのシステムコールの回数" を表す). シングルプロセス時の性能は 31.0 [Trans/sec] であり, これは往復遅延時間 0.032 秒の逆数とほぼ等しく期待通りの性能が得られている. 同様に, 2 プロセス並列時の性能も 62.0 [Trans/sec] となり 2 プロセス合計でほぼ 2 倍の性能が得られていることが確認された. これらに対し並列プロセス数を 3 以上に向上させても合計性能は 2 並列時と同程度となり, 多

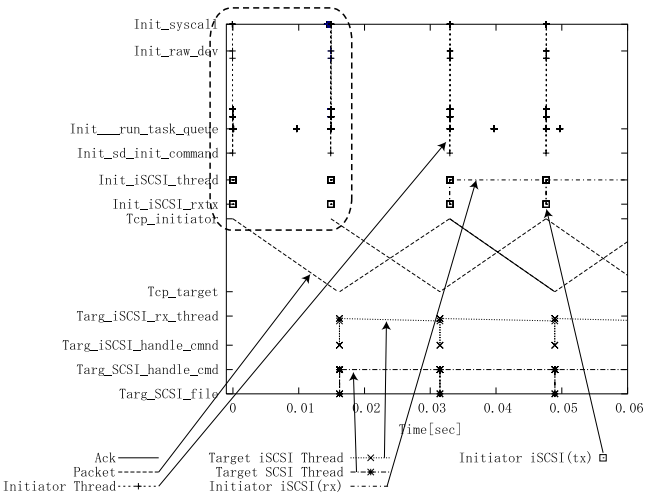

図 4　並列アクセスのトレース図 (default)：A

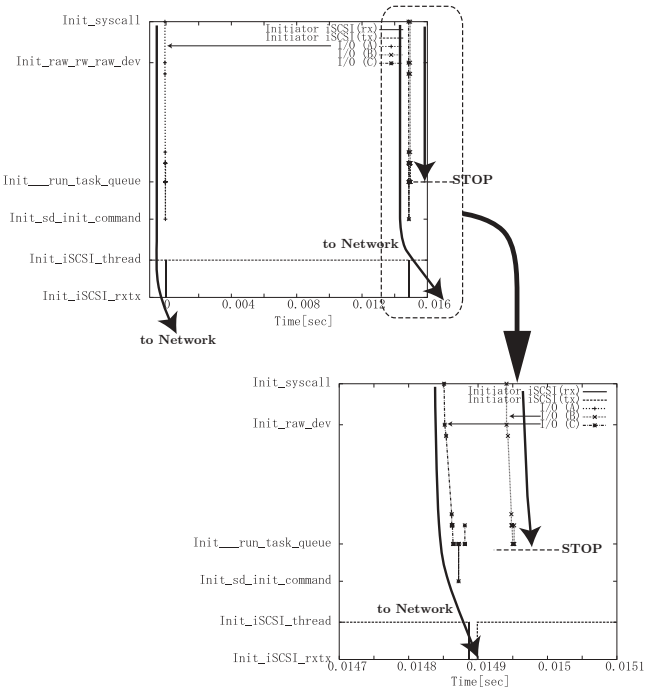

図 5　並列アクセスのトレース図 (default)：B

段プロトコルスタックで構成される iSCSI ストレージアクセスのいずれかの層で並列動作数を 2 に制限していると考えられる.

3.3　並列アクセスのトレース

前節の 3 プロセス並列アクセス実験のトレースを可視化し図 4 を得た. "Initiator Thread" が切断して記してあるのは OS のプロセススケジューラのコンテキストスイッチが発生しプロセスが停止/再開したことを表している (同様の理由で 0.010 秒, 0.040 秒などにおいてトレースがプロットされている). 同図よりイニシエータ (サーバ計算機) からターゲット (ストレージ機器) に対し 1 往復時間に内に同時に 2 個の I/O 要求しか送出されておらず, 並列度制限はイニシエータ側に存在すると予想される.

次に並列度を制限している部分の巨視的な解析を行う. 図 4 左上波線部を拡大し可視化したものを 図 5 の左上に記す. そ

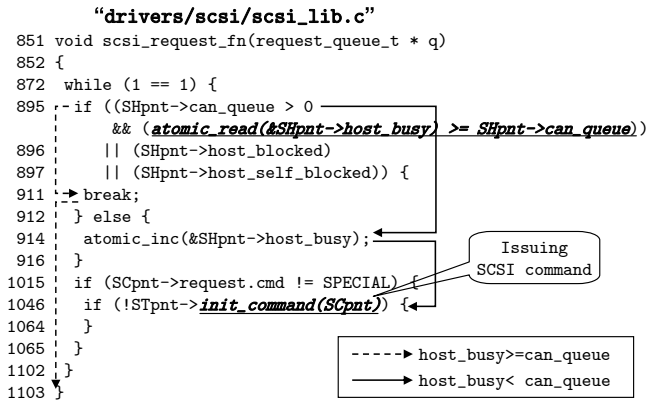

図 6　Linux SCSI 層のトレース: "drivers/scsi/scsi_lib.c"

して, その波線部の拡大図を同図右下に記す. 同図では, 並列して動作している 3 個の I/O 要求を "I/O(A)", "I/O(B)", "I/O(C)" と別の線として記した.

図 5 左上より I/O(A), (B), (C) は順に時刻 0.000 秒, 0.015 秒, 0.015 秒にシステムコールを発行していることが確認できる. ターゲットからの応答を待たずに 1 往復時間 (32ms) 内に 3 個のシステムコールが発行されていることからシステムコールの発行において並列性が制限されていないことが分かる. また, "I/O(A)" のトレースより "I/O(A)" の要求は raw デバイス層, SCSI 層, iSCSI 層, TCP/IP 層を経由し実際にネットワークに送出されていることも確認できる. 次に, 同図右下より "I/O(C)" のシステムコールは時刻 0.01485 秒に発行され, 各層を経由し要求はネットワークに送出されていることが確認できる. これに対し "I/O(B)" のシステムコールは時刻 0.01494 秒に発行され, raw デバイス層を経由し raw デバイスによる命令は発行されているが SCSI 層における SCSI 命令の発行に到っていないことが確認でき, SCSI 命令の同時発行上限が 2 となっていると予想できる.

次に, 並列度制限箇所の微視的な解析を行う. SCSI 命令が即時に発行される最初の 2 要求 (I/O(A), I/O(C)) と命令の発行が拒否される 3 個目の要求 (I/O(B)) のトレース分岐点は SCSI 層である図 6 の Linux カーネルの "drivers/scsi/scsi_lib.c" 部である. 同実装箇所は現在アクティブ (処理途中) である命令数 host_busy(注1) と, 下位層 (本例では iSCSI ドライバ) が同時に受け付け可能なアクティブな SCSI 命令数 can_queue(注2) の比較部となっている. UNH iSCSI 実装において "can_queue" の初期値は 2 となっており, アクティブ命令数 host_busy は 0 から開始される. 最初の 2 要求 (I/O(A),I/O(C)) のトレースでは host_busy はそれぞれ 0, 1 となっており図 6 における "host_busy<can_queue" に示される動作が記録された. すなわち, 同図 914 行目において host_busy をインクリメントし, 1046 行目において SCSI 命令が発行される動作が記録された. 3 個目の要求 (I/O(B)) においては host_busy が 2 であり "host_busy>=can_queue" に示される様に SCSI 命令が発行されない動作が記録された. 以上の微視的なトレース解析により, 並列度を制限し性能向上を妨げている原因は iSCSI ドライバにおける "can_queue" の値が不十分であることだと考えられる.

(注1): Linux "drivers/scsi/hosts.h" にて "commands actually active on low-level" と解説されている

(注2): UNH iSCSI 実装の "initiator/iscsi_initiator.c" にて "max no. of simultaneously active SCSI commands driver can accept" と解説されている

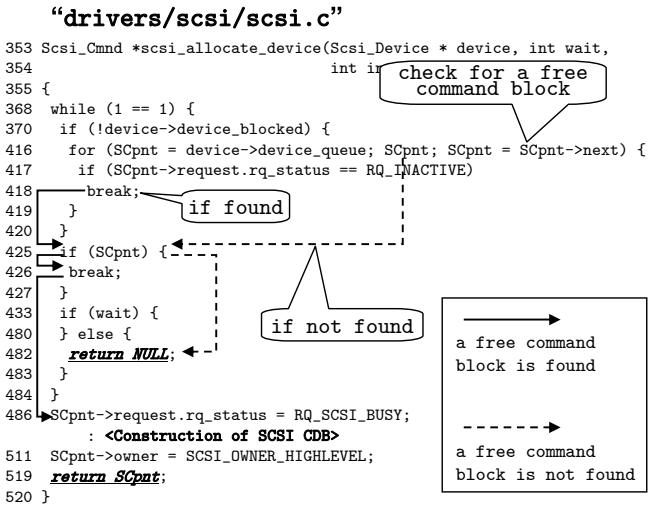

図 7　Linux SCSI 層のトレース: "drivers/scsi/scsi.c"

3.4　発見された問題点の回避

前節の解析から高遅延環境における並列アクセスの性能を向上させるには "can_queue" の値を増加させることが重要であると考えられる. これを十分大きな値 (後述の理由により 8 以上) である 30 に増加させ第 3.1 節の実験を行い図 3 の "can_queue=30, queue_depth=8" の実験結果を得た. 同結果より, 全プロセスの合計性能は並列度 8 までは線形に増加させることが可能となったことが確認できる.

3.5　問題回避後の再解析

次に, 並列度が 8 に制限されている原因について考察する. 第 3.3 節同様に巨視的トレース解析を行うと, 制限箇所は同様に SCSI 層であることが確認された. また, 命令が実際に送出される最初の 8 要求と, 送出されない 9 個目の要求のトレースの分岐点は, Linux SCSI 層実装 "drivers/scsi/scsi.c" の void scsi_allocate_device() 部であった. その詳細を図 7 に記す. 同図は SCSI 命令ブロック作成部であり, 確保されているブロックキューから未使用ブロックを検索しそのブロックに SCSI 命令を格納し, それを返す関数である. 最初の 8 要求では "a free command block is found" に示されるトレース結果が記録された. すなわち, 416 行目において未使用ブロックを検索し, 417 行目においてそれを発見している (よって変数 SCpnt には発見されたブロックへのポインタが納められる). そして, SCSI 命令ブロック作成部 (同ファイル 486 から 511 行目) に移動し同関数は作成した SCSI 命令を返している. これに対し, 命令が発行されない 9 個目の要求では, "a free command block is not found" に示されるトレース結果が記録された. すなわち, 416 行目において未使用ブロックを検索したが発見されず (これにより変数 SCpnt は NULL となる), 482 行目において NULL を返している. 同関数は, "drivers/scsi/scsi_lib.c"

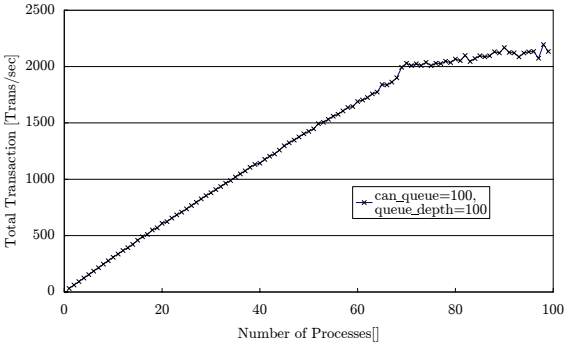

図 8　実験結果 B：並列 I/O の合計性能

内の関数 `scsi_request_fn()` から呼び出されており, SCSI 命令ブロックが返された場合は SCSI 命令を発行し, `NULL` が返された場合は SCSI 命令を発行しないよう実装されている.

SCSI 命令キュー長 (Linux SCSI 実装内では “`queue_depth`” と呼ばれる) は SCSI 層の下位層により指定され, 今回の例では iSCSI 層の実装 (UNH iSCSI) の初期値 8 が使用された.

3.6　再解析に発見された問題の回避

前節の解析により “`can_queue`” 増加後の並列度制限原因は “`queue_depth`” であると考えられる. そこで, “`can_queue`”, “`queue_depth`” をともに十分大きな値 (後述の理由により 69 以上) である 100 とし再度性能を測定し, 図 8 の結果を得た. 図より, 発見された並列度制限要因の解決により並列度はさらに上昇し, 並列度 69 まで合計性能はほぼ線形に上昇することが確認された. 並列度 69 における合計性能は 1992.9[Trans/sec] であり, 並列度 98 における合計性能は 2195.0[Trans/sec] であったため, 提案システム適用前の最適化を行っていない状態 (最大で 61.9[Trans/sec]) と比較し合計性能は 32.2 倍や 35.5 倍に向上されたこととなる.

このように, 提案トレースシステムを用いて IP-SAN システムの振る舞いを観察することにより性能劣化原因を的確に発見することが可能であり, それらの解決により性能を大きく向上させることが可能であることが確認された.

4.　IP-SAN システムと応用の統合考察

前章までに, IP-SAN システムの振る舞いの観察について述べてきた. ユーザ空間のアプリケーションの要求を受けて動作する IP-SAN システムは, 要求を発行する応用プログラムと併せて考察することによりさらに詳細に振る舞いの観察を行うことが可能となり, より適切な応用プログラムの考察が可能になると考えられる. また, より現実的なアプリケーションの考察としてファイルシステムを含む IP-SAN トレースシステムが重要と考えられる. 本章では, 応用プログラムとファイルシステムを含む IP-SAN システムを統合的に観察する手法を提案し, 現時点において作成されている試作システムの紹介を行い, その有効性を考察する.

4.1　ファイルシステム解析システム

第 2. 章で紹介したトレースシステム同様に, オープンソース OS である Linux のファイルシステムのソースコードにモ

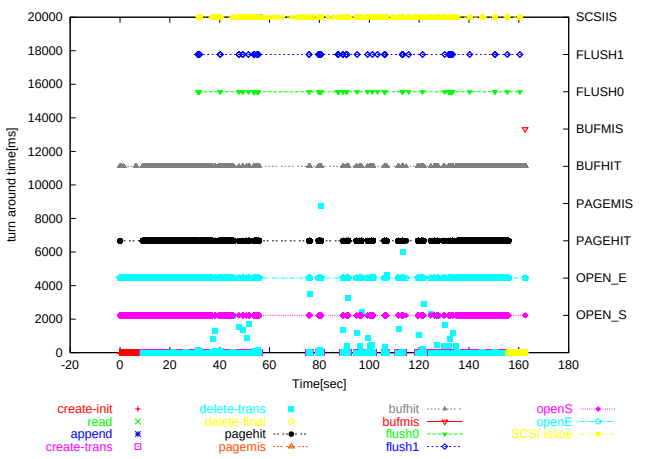

図 9　PostMark 実行の解析 (A)

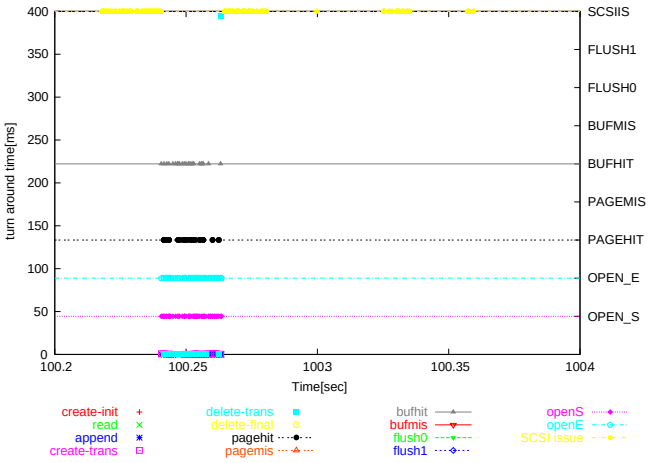

図 10　PostMark 実行の解析 (B)

ニタ用のコードを追加することによりファイルシステムトレースシステムを構築した. ファイルシステム実装としては, ext2 (Linux 2.4.18) を採用した. ファイルシステムの各種操作の動作の記録や, バッファキャッシュ/ページキャッシュの振る舞いの記録を残し観察できるシステムを構築した.

このトレースシステム上でファイルシステムを扱うベンチマークソフトウェア PostMark [14] を実行し, その振る舞いを観察した例を以下に示す. PostMark は代表的な I/O 処理の性能を測定するベンチマークの一つである. インターネットサーバ (E-Mail, NetNews, web-based commerce などを想定) の様な短命でサイズの小さいファイルを多数扱うプログラムの性能を評価することを目的としている. 同ベンチマークは, 主に以下に示す処理を行う. ① 初期化処理として, 初期ファイルとして指定個数のファイルを作成する, ② トランザクション性能の測定として, 指定された回数だけ後述のファイル操作処理を反復する, ③ 終了処理として, ベンチマーク中に作成したファイルを全て削除する. また, ②のファイル操作処理としては, ファイルの作成または削除, ファイルの読込または追記, を指定回数反復する. 上記 PostMark を以下の設定において実行した: 初期ファイル数 25,000 個, ファイル操作 100,000 回, ファイルサイズ 500Byte〜10,000Byte, I/O 対象 iSCSI 接続ディスク, Read/Write ブロックサイズ 512Byte. iSCSI ディスクは同様

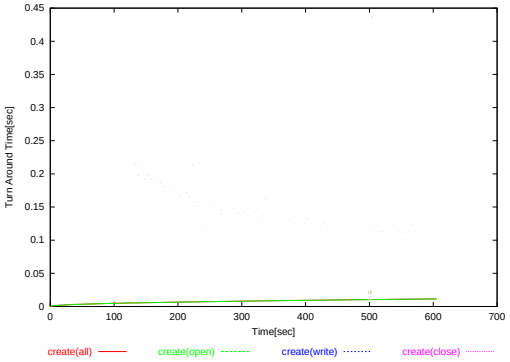

図 11　ファイル作成実験 A

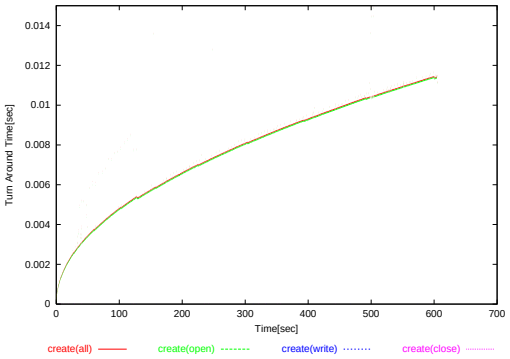

図 12　ファイル作成実験 B

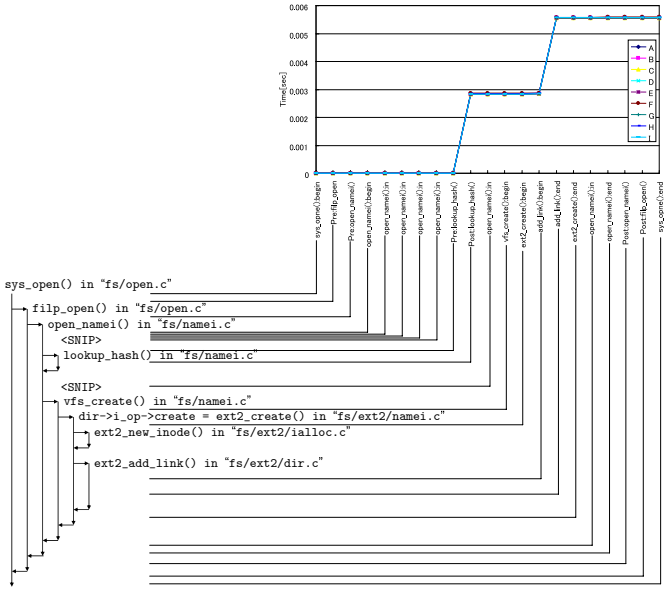

図 13　ファイル作成操作

に ファイルモードを用い, イニシエータ - ターゲット間の片道遅延時間は 8m 秒とした. 同実験を提案システムにより観察することにより図 9, 図 10 の様な解析結果を得ることができる. 両図左縦軸は各操作 (初期化ファイル作成, トランザクション処理, 終了処理ファイル削除) のターンアラウンドタイムを表し, 右縦軸は, 各種イベントの発生時刻を表しており, 下から順に ファイルオープン開始, ファイルオープン終了, ページキャッシュヒット, ページキャッシュミスヒット, バッファキャッシュヒット, バッファキャッシュミスヒット, ディスクキャッシュのフラッシュ(1), ディスクキャッシュのフラッシュ(2), SCSI コマンドの発行を表している. 同図より, 同アプリケーションの同実行例においてはファイルシステムのキャッシュがヒットしており, 多くの場合は各操作のターンアラウンドタイムはネットワーク往復時間 (16m 秒) より十分に小さいく各処理はネットワークの往復を伴っていないことが確認できる. また, 図 9 からファイル削除処理のみ著しく時間を要することがある (最長の例において約 20 秒) ことなどが確認できる. 図 10 より, ファイル削除要求発生後に SCSI コマンドが発行され (これは iSCSI コマンドの送受信の発生を意味する), 削除処理に多くの時間を要している様子が確認できる. このように提案システムを用いることによりアプリケーションやファイルシステム (キャッシュなど) を含む IP-SAN システムの振る舞いの観察が可能となる.

4.2　ファイル操作の解析例

本節では, 試作したファイルシステム解析システムを用いて, ファイル操作を行うアプリケーションおよび IP-SAN システムの振る舞いの解析を行い, その性能に対する解説を行い, 当システムが試作段階ながら有効であることを示す. Initiator-Target 間人工遅延時間以外を第 3.1 節と同様の実験環境 (同一計算機, 同一モード (ファイルモード) を使用) とし, Initiator-Target の両機をクロスケーブルで接続 (片道遅延時間は約 0.08m 秒程度となる) した環境においてファイル作成操作を繰り返すベンチマークプログラムを動作させ, その振る舞いを観察した. Initiator-Target 間の遅延時間を大きくするとディスクへの遅延書き込みのフラッシュ時の影響が大きくなりより複雑な例となるが, 本節では単純な例を紹介する. 実験内容は以下の通りである. 上記環境における iSCSI 接続のディスクのパーティションを ext2 ファイルシステムでフォーマットし, ファイルシステムを構築する. 同ファイルシステム上にディレクトリを作成する. 次に, 同ディレクトリ内に, 16KB のファイルを 100,000 個新規に作成する. ファイルの作成は, システムコール “`fopen()`” でファイルを新規にオープンし, “`fwrtie()`” により 16KB 書き込み, “`fclose()`” によりファイルを閉じることにより行う. 上記処理を行いベンチマークプログラム内において各処理に要した時間を計測し, 図 11, 12 の結果を得た. 両図は縦軸の範囲を変えたのみの同一のグラフであり, 図 11 が全体を表示しており, 図 12 が一部の拡大表示である. 両図縦軸は各操作のターンアラウンドタイムであり, 横軸は時刻である. 前述のように本例においてはファイル作成操作は,“fopen()”, “fwrite()”, “fclose()” の 3 操作から構成されており, 同図内の “create(open)”, “create(write)”, “create(close)” がそれぞれに要した時間を表している. また, “create(all)” がファイル作成

全操作に要した時間であり, 上記の 3 操作 (open, write, close) の合計である.

まず, 両図より多くの例において作成操作 (all) は, 0.012[秒] 以下で終了しているが, 図 11 より 0.1[秒] 以上の時間を要した操作が少数存在していること (本例に置いては 100,000 回のファイル作成を行い, 0.1[秒] 以上要した例が 82 回確認された) が確認できる.

次に, 図 12 より, 作成済みファイルの増加にともないファイル作成操作ターンアラウンドタイムは基本的には増加している (増加の速度はほぼ平方根に比例) こと, 最終的には (100,000 個目付近では) ファイル 1 個の作成に 10[m 秒] 以上の時間を要していることが確認できる. これは, ネットワークの往復時間 (0.16m 秒程度) と比べ十分に大きい時間であると言える.

またファイル作成操作は open, write, close の 3 操作より構成されているが, 図 12 よりファイル作成の操作の全時間 (all) と open 処理に要している時間 (open) がほぼ等しく, ファイル作成に要する時間のほとんどが open 処理に消費されていること (同図において write 処理は 平均 12[u 秒], 最大 217[u 秒]. close 処理は 平均 21[u 秒], 最大 191[u 秒] である) が確認できる. 以上がベンチマークアプリケーションを用いてユーザ空間から測定した性能である.

さらに以下で, 試作した解析システムによるカーネル空間内部のファイルシステムの振る舞いの解析結果を紹介する. ext2 ファイルシステムでは, ファイルオープンによる新規ファイルの作成は, 図 13 左下の様な手続きにより行われる. 同図左下は, Linux 実装のファイルシステムの実装における関数呼び出しの推移を記した模式図である. すなわち, ファイルオープン処理は, Linux カーネル実装 "`fs/open.c`" 内の関数 `sys_open()` より始まり, 同関数内からさらに, `filp_open()` 関数が呼び出され, さらに `open_namei()` 関数が呼び出されていくこととなる. 次に, 同図のこれらの処理の進行の記録を時間軸上に表示したグラフを同図右上に示す. 同図右上の横軸は線で結ばれた模式図内 (左下) の各通過点を表しており, 縦軸は各通貨点を通過したときの時刻である. ただし, 時刻は各ファイル作成開始時刻からの経過時間である. 同図右上には, A〜I までの 9 個の進行の例が示されているが, これらは図 11, 12 の時刻 144[秒] における連続する 9 個の例であり, 図 12, 13 から分かるように, この時点におけるファイル作成に要する時間は 5.6[m 秒] 程度である.

同図右上より関数 "`lookup_hash()`" 直前の通過時刻と同関数直後の通過時刻に大きな差 (約 2.8[m 秒]) があることや, "`add_link()`" 関数の前半通過時刻と, 後半通過時刻に大きな差 (約 2.7[m 秒]) があることが確認され, 上記 2 処理がファイル作成操作の所要時間のほとんどを占めていると結論づけられる. 前者 (lookup_hash) は, dentry(Linux ファイルシステムにおける i ノードのディレクトリへの登録) の検索であり, 後者 (ext2_add_link) も同様に `ext2_add_link()` 関数内においてdentry を検索する処理である. 両者とも同ファイルシステムでは線形検索として実装されているため, 検索時間は既存ファイル数に比例する. このため, n 個目のファイルを作成するのに要する時間は n に比例し, 1 個目から n 個目までの合計 n 個のファイルを作成するのに要する時間は $\sum_1^n k$ に比例することとなる. $\sum_1^n k$ が n^2 に比例するため, "n 個のファイルを作成するのに必要な時間" は n^2 に比例することとなる. これに対し n 個目のファイルを 1 個作成するのに要する時間は n に比例する. 結果として, "ファイル作成した時刻" と "ファイル作成に要した時間" の関係は図 12 の様な平方根に近いグラフとなることが理解できる.

現試作実装では性能向上等の考察には到らないが, リードキャッシュのヒット/ミスヒットや遅延書込のフラッシュなどの挙動の把握がある程度なされ, 各種操作内の各処理のどの程度の時間を要しているかを定量的に観察することが可能となっている. 本章で紹介した例では, ファイルシステム内の検索演算処理 (10[m 秒] 程度要している) がネットワーク遅延よりも大きな時間を要し性能に支配的な影響を与えていることを定量的に観察することができた. さらに詳細な観察が可能となるよう実装を進めることにより, ネットワークストレージシステムの性能を考察する上で有用な手法になると期待できる.

5. 関 連 研 究

iSCSI を用いた IP-SAN の性能の評価に関する研究としては, 文献 [1], [3], [4], [15]〜[18] があげられる. 文献 [1] は早期に SCSI over IP の性能評価を行った開拓的な研究である. Sarkar らは文献 [3], [4] において CPU 使用率に着目し iSCSI アクセスの性能についての評価を行い, iSCSI アクセスにおける CPU 使用率の増加の程度や, TOE(TCP Offload Engine) の貢献の限界などを示している. 文献 [16] は, iSCSI, SMB, NFS の性能評価と比較を行っている. 文献 [15] は iSCSI 性能を評価しソフトウェア処理手法は FC-SAN に匹敵する性能を提供できることを示している. 文献 [17] は, IP 接続ストレージのアクセス手法として iSCSI と NFS の比較を行い, NFS に対する iSCSI の優位性等を示している. 文献 [18] は, IPsec や SSL を用いる iSCSI の性能を評価している. 以上の研究は各種状況における iSCSI 性能の評価を行ったものであり, iSCSI の性能を知る上で有用な研究であると言える. しかし, システムの外部から負荷を与え性能を評価したものでありシステム内部の振る舞いについて考察を行ったものではない. また, 性能の向上方法について十分な考察を行ったものではなく, 性能を向上させるには適切な能力の計算資源を用意する以外の方法を提供しない. よって, 性能向上方法の提供を目指す本研究はこれら既存の研究に対しても十分に有用であると考える.

藤田らの文献 [5] は, iSCSI ターゲットの内部の実装手法も考慮して iSCSI 性能の評価を行い, OS のカーネルに変更を加える手法や低レベルインターフェイスを使用する実装手法が性能において優れていることを指摘してる. ターゲット実装に着目し詳細な考察を行っている点において, システム全体の考察を目指す我々の研究と目的が同じでは無いが, ターゲットシステム実装の詳細な考察を行った既存の研究として価値が高いと思われる.

6. おわりに

本稿では, サーバ計算機とストレージ機器の分散協調システムとして動作する IP-SAN システムのアクセストレースシステムを提案し, その有効性の検証を行った. その結果, 提案システムは多段プロトコルスタックで構成される分散システム IP-SAN の振る舞いの把握を容易にし, 的確に性能劣化原因を発見することが可能であった. 本稿の例では多並列アクセス時に 30 倍以上の性能向上が実現され, 提案手法が IP-SAN システムの性能向上の実現に有効な手法であることが確認された. さらに, アプリケーションやファイルシステムを含め統合して解析するする手法の提案を行い, 試作システムの動作例を紹介した. 同システムを用いることにより, キャッシュのフラッシュなどの振る舞いが性能に大きな影響を与えている様子や, ファイルシステム内の検索処理などが性能に支配的な影響を与えている様子を定量的に観察することが可能であることが確認された.

今後は試作したファイルシステムのトレースシステムの実装を進め, 各種ファイル操作の詳細な記録の採取を可能とし, 実際のアプリケーションの振る舞いの観察や性能向上を実現していく予定である.

文　献

[1] Wee Teck Ng et al. “Performance Evaluation and Improving of Sequential Storage Access using iSCSI Protocol in Long-delayed High throughtput Network”. In *Proc. of IEICE The 14th Data Engineering Workshop*, March 2003.

[2] F. Neema and D. Waid. “Data Storage Trend”. In *UNIX Review, 17(7)*, June 1999.

[3] Prasenjit Sarkar, Sandeep Uttamchandani, and Kaladhar Voruganti. “Storage over IP: When Does Hardware Support help?”. In *Proc. FAST 2003, USENIX Conference on File and Storage Technologies*, March 2003.

[4] Prasenjit Sarkar and Kaladhar Voruganti. “IP Storage: The Challenge Ahead”. In *Proc. of Tenth NASA Goddard Conference on Mass Storage Systems and Technologies*, April 2002.

[5] 藤田智成 小河原成哲. “iSCSI ターゲットソフトウェアの解析”. In **先進的計算基盤システムシンポジウム** *SACSIS 2004*, May 2004.

[6] “IETF IPS”. `http://www.ietf.org/html.charters/ips-charter.html` .

[7] J. Satran et al. “Internet Small Computer Systems Interface (iSCSI)”. `http://www.ietf.org/rfc/rfc3720.txt` , April 2004.

[8] IETF Home Page. `http://www.ietf.org/` .

[9] 喜連川優 山口実靖 小口正人. “iSCSI ストレージアクセスのトレースシステム”. In **夏のワークショップ** *DBWS 2004* **電子情報通信学会技術研究報告データ工学 信学技報** *Vol.104 No.177*, July 2004.

[10] 山口実靖 小口正人 喜連川優. “iSCSI 解析システムの構築と高遅延環境におけるシーケンシャルアクセスの性能向上に関する考察”. **電子情報通信学会論文誌** *D-1*, 87, February 2004.

[11] University of new hampshire interoperability lab. `http://www.iol.unh.edu/` .

[12] iSCSI reference implementation. `http://www.iol.unh.edu/consortiums/iscsi/downloads.html` .

[13] L. Rizzo. dummynet. `http://info.iet.unipi.it/~luigi/ip_dummynet/` .

[14] PostMark. `http://www.netapp.com/tech_library/postmark.html` .

[15] Stephen Aiken, Dirk Grunwald, and Andy Pleszkun. “A Performance Analysis of the iSCSI Protocol”. In *IEEE/NASA MSST2003 Twentieth IEEE/Eleventh NASA Goddard Conference on Mass Storage Systems and Technologies*, April 2003.

[16] Yingping Lu and David H. C. Du. “Performance Study of iSCSI-Based Storage Subsystems”. *IEEE Communications Magazine*, August 2003.

[17] Peter Radkov, Li Yin, Pawan Goyal, Prasenjit Sarkar, and Prashant Shenoy. “A performance Comparison of NFS and iSCSI for IP-Networked Storage”. In *Proc. FAST 2004, USENIX Conference on File and Storage Technologies*, March 2004.

[18] Shuang-Yu Tang, Ying-Ping Lu, and David H. C. Du. “Performance Study of Software-Based iSCSI Security”. In *1st International IEEE Security in Storage Workshop*, December 2002.