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

2009－ARC－182 (7)
2009－HPC－119 (7)
2009／2／26

ファイルステージング再考：オンデマンド化と高速化に向けたプロトタイプ実装の評価

堀 敦史[†] 鴨志田 良和[†] 松葉 浩也[†]
安井 隆[††] 住元 真司[†††] 石川 裕[†]

本稿では，新しいファイルステージングシステム，Catwalk，を提案する．Catwalk ではファイルステージングはオンデマンド化され，ファイルのステージインはより高いバンド幅が得られるようパイプライン処理により高速化されている．またオンデマンド化によりユーザはファイルステージングの記述が不要になる．プロトタイプが実装され評価された．評価の結果から，Catwalk は計算ノードに比例するステージインバンド幅が実現された．計算ノードが 16 の場合，NFS の 2 倍以上の総バンド幅を得ることができた．Catwalk の方がファイルアクセス数が多いにも関わらず，Catwalk のステージアウトバンド幅は NFS とほぼ同様の性能が実現できた．

Rethinking File Staging: Evaluation of On-Demand File Staging Prototype System

Atsushi Hori,[†] Yoshikazu Kamoshida,[†] Hiroya Matsuba,[†]
Takashi Yasui,[††] Shinji Sumimoto[†††] and Yutaka Ishikawa[†]

In this paper, a new file staging system, Catwalk, is proposed. In Catwalk, file staging is triggered in an on-demand way and stage-in process is pipelined so that it can achieve higher bandwidth. The on-demand file staging frees users from writing file staging scripts. A prototype system is designed and implemented. The evaluation of Catwalk indicates that it can exhibit the stage-in bandwidths proportional to the number of compute nodes, more than 2 times fatesr than NFS when the number of nodes is 16. Despite the fact of larger number of file accesses, the stage-out bandwidths of Catwalk are comparable with the ones of NFS.

はじめに

ファイルステージングは，計算に必要な「遠く」にあるデータを「近く」にコピーする，あるいは，計算の途中あるいは最終結果のデータを「遠く」にコピーする手法であり，古くはメインフレームの時代から良く知られている．最近の高性能計算機はクラスタとなる場合が多く，ファイルサーバとクラスタを構成する計算ノードの間でファイルステージングを用いている場合も多い．

ファイルステージングでは，ステージングの対象となるファイルと方向を事前に陽に明記するする必要があるため，記述のミスはジョブ実行の失敗となる．特にバッチスケジューリング下で，長い間待たされた後に記述ミスが発覚することも多く，ユーザの立場からはジョブの再投入による応答時間の遅さ，運営の立場からは無効なジョブの実行による実質的な効率の低下といった問題がある．b 仮にジョブの実行時にオンデマンドでファイルステージングが実現できたとすると，ステージングの記述が不要になると同時に記述ミスを無くすことができる．

一方，大規模なクラスタでは，ファイルステージングではなく並列ファイルシステムを用いる場合があるが，これにはそれなりのハードウェア投資と知識が必要であり，試験的に導入するのも簡単ではない．また，センター運用されているクラスタでは，全ユーザのジョブを止めてまでして試験導入するのは難しい．このため root 権限が不要なインストールや実行が可能なシステムの方が望まれる．

本稿では，クラスタを対象としたオンデマンドのファイルステージング方式を提案するものである．実装されたシステムは，Catwalk と名付けられ，インストールや実行に際し root 権限を全く必要としない．Catwalk は T2K オープンスパコン（東大）上で評価

† 東京大学
The University of Tokyo
†† 日立製作所
Hitachi, Ltd.
††† 富士通研究所
Fujistu Laboratory

され，広く使われている NFS を用いた並列ファイルアクセス性能が比較された．

1. 設計と実装

クラスタにおけるファイルステージングでは，ユーザジョブからのファイルアクセスは全て計算ノードに付随するローカルディスクが対象となる．以下に，Catwalk の基本設計方針と実装の概要について述べる．

1.1 基本設計方針

Catwalk は並列ファイルシステムを持たないクラスタに容易に導入可能であることを設計の前提とし，既にファイルステージングを導入しているクラスタや，NFS による並列ファイルアクセス性能に不満があるクラスタを想定している．より具体的な設計方針としては，1) インストールあるいはソースからのビルド時に root 権限が不要であること，2) もっとも普及している Ethernet を用いること，3) 新たなハードウェア投資なしに既存のクラスタ上で動作可能なこと，の 3 点が挙げられる．1) の条件から，Linux で標準的にインストールされない，常駐するようなデーモンやカーネルドライバは用いることができなし，Catwalk 独自のの常駐デーモンやカーネルドライバも不可となる．また 2) からは，1 Gbps 程度のバンド幅のネットワークを前提に設計することになる．3) からは，バンド幅を稼ぐための複数のファイルサーバやファイルサーバと計算ノード間に Ethernet よりも高速なネットワークの存在を仮定しない．

Catwalk では基本的に 1 台のファイルサーバしか仮定しない．このため特殊な並列ファイルシステムを必要とせず，しばしばボトルネックの原因となるメタデータの管理も不要である．さらに 1 台のファイルサーバはバックアップや管理運営が容易，という利点もある．

これらの条件は，一般的なクラスタとして最低限の要件であり，特殊なハードウェアやソフトウェアを仮定していないことを意味する．本研究は普通のクラスタでいかに NFS よりも高いファイル性能を引き出すか，という点にチャレンジするものでもある．

1.2 オンデマンド化の実装

通常のファイルステージングでは，ジョブの実行に先立ち，必要となるファイルを計算ノードのローカルディスクにステージインし，ジョブの終了時に必要なファイルをファイルサーバにステージアウトする．ステージングのオンデマンド化では，実行時ライブラリにある `open()` などの関数にフックを掛けることで，1) ステージングの対象となるファイル，2) I/O の方向（入力／出力），を知ることができる．Linux では `LD_PRELOAD` 環境変数に動的ライブラリを設定することで，任意の glibc 関数にフックを掛けることが可能になっている．本システムではこの `LD_PRELOAD` 環境変数にフックのためのライブラリを自動的に設定するようにした．

Catwalk では，読込 `open()` のフックにより，ステージインに必要なファイルの情報を得る．同様に書込 `open()` あるいは `creat()` のフックにより，ステージアウトの情報を得る．現時点の Catwalk では，上記以外に `fopen()`，`stat()` や `exec()` の類に対しフックが設けられており，これらは基本的にステージインとして扱われる．Catwalk ではそれ専用の API を特に必要としないし，NFS やファイルステージングを仮定した並列プログラムならばプログラムに手を入れる必要は全くない．

1.3 ステージングの実装

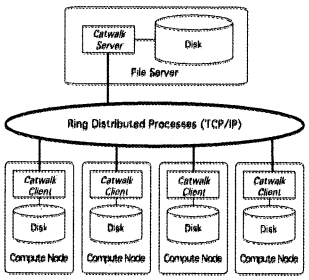

図 1 リング分散プロセス構造

Catwalk は図 1 に示すようなリング状の分散プロセス構造を持つ．ステージングの元となるファイルサーバおよび計算ノード上にそれぞれひとつの Catwalk プロセスが生成される．隣接するプロセスは TCP/IP により通信が可能になっている．これらのプロセスはユーザジョブの起動とともに生成され，ジョブの終了とともに消滅する．以下の説明で，ファイルサーバ上の Catwalk プロセスを「(Catwalk) サーバプロセス」．計算ノード上の Catwalk プロセスを「(Catwalk) クライアントプロセス」と呼ぶ．

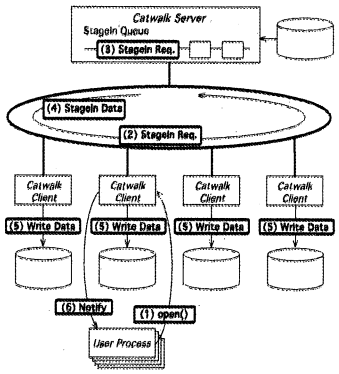

図 2 ステージイン手順

ステージイン処理の概略を図 2 に示す．

(1) ユーザプロセスで open() が呼ばれるとフック関数が起動し，それが読込オープンで会った場合，ステージインに必要な情報を同じノード上のクライアントプロセスに通知し，フック関数はクライアントからの返答を待つ．

(2) クライアントプロセスは，この要求を次のプロセスに渡す．この要求はリングに沿って順次転送され，最終的にサーバプロセスに到達する．

(3) サーバプロセスは受け取ったステージイン要求を StageIn Queue に入れ，このキューの先頭から順次ステージイン要求が処理される．

(4) サーバプロセスはステージイン要求に従い該当するファイルをオープンし，そのファイル属性およびファイルの内容を次のクライアントプロセスに渡す．

(5) ステージインファイルのデータを受け取ったクライアントは，該当するファイルをオープンし，データを書き込み，隣のプロセスにリレーする．この処理は全てのクライアントで行われるため，結果として全ての計算ノード上にコピーが生成されることになる．

(6) サーバのファイル読込が EOF に達すると，そのが各クライアントに通知され，ステージイン要求を出したユーザプロセスに完了結果を通知する．ユーザプロセスでは結果を受け取り，ステージインが正常に終了したならば，glibc の open() を呼び，open() のフック関数から戻る．

Catwalk をリング状の分散プロセス構造としたのは，1) ファイルのブロードキャストをパイプライン処理し，バンド幅を稼ぐ，2) 逐次的な処理のためプログラミングが比較的容易，という理由からである．このようなパイプライン処理では計算ノード数に比例したパイプライン段数となり，ファイルサーバからのデータ出力バンド幅が一定にも関わらず，理論上はファイルが十分に大きければ計算ノード数に比例したバンド幅が得られるという利点がある．

この方式で評価を開始すると，計算ノード数が多い程，また，ステージインの対象となるファイルの大きさが大きい程，処理時間にかなりのバラツキがでる現象が確認された．これは，計算ノードで発生した Linux のバッファキャッシュのフラッシュによりパイプラインのストールが発生しているものと推測される．このフラッシュはキャッシュの状態に応じて発生するが，各計算ノードのキャッシュの状態は一様ではないため，キャッシュのフラッシュは時空間的にランダムに発生すると考えられる．ストールから復帰したとしても別なノードで再度ストールが発生するという状態が続く．このようにして計算ノードのあちこちでランダムで発生するパイプラインのストールは全体のスループットを著しく低下させる．

この影響を回避するために，計算ノード上のステージインで書込むファイルを O_DIRECT フラグでオープンした．O_DIRECT でオープンされたファイルにおける I/O はバッファキャッシュを経由しない．しかしながら，O_DIRECT フラグによるファイルのアクセスはキャッシュの効果が失われるため I/O 速度が低下するという悪影響もある．

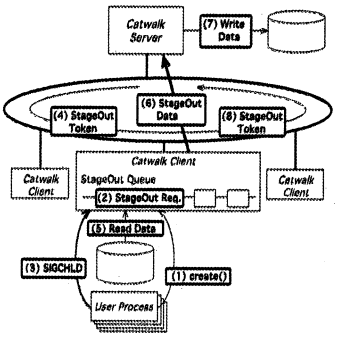

図 3 ステージアウト手順

ステージアウト場合の処理の概略を図 2 および以下に示す．

(1) ステージイン同様，ユーザプロセスで open() が呼ばれるとフック関数が起動し，それが書込オープンであった場合，ステージアウトに必要な情報を同じノード上のクライアントプロセスに通知する．

(2) Catwalk クライアントプロセスでは，ステージアウト要求を StageOut Queue に入れ，ユーザプロセスの終了を待つ．

(3) 計算ノードの全てのユーザプロセスが終了すると，その旨をサーバプロセスに通知する．サーバプロセスでは全てのクライアントプロセスからのプロセス終了通知を待つ．

(4) サーバプロセスが全てのユーザプロセスの終了，すなわちジョブの終了を感知すると，最初のクライアントプロセスに StageOut Token を渡す．

(5) StageOut Token を受け取ったクライアントプロセスは，StageOut Queue にあるステージアウト要求を順序処理する．処理に先立ち，サーバプロセスに直接，新たな TCP/IP 接続を生成し，以後，このクライアントからのステージアウトの情報は全てこの TCP/IP 接続を経由する．ステージアウト要求のあったファイルがユーザプロセスによって消されていないならば，そのファイルをオープンし，ファイルの属性とファイルの内容をサーバに送る．

(6) サーバプロセスではステージアウトの情報を受け取り，該当するファイルをオープンし，受け

取ったファイルの内容を書き出す．

(7) StageOut Queue が空になると，StageOut Token を隣のクライアントに渡し，ステージアウト処理は隣のクライアントに引き継がれる．この時，引き継ぎ先がサーバプロセスであった場合は，計算ノード上の Catwalk の全てのプロセスを終了待ってサーバ上の Catwalk プロセスが終了する．

2. 評　　価

表 1 に示す 17 ノード（NFS サーバ 1 台と計算ノード 16 台）から構成されるクラスタを用いた．本クラスタには 1 Gbps の Ethernet と並列計算用の Myrinet 10G の 2 種類のネットワークがあるが，今回の計測では Ethernet がステージングに関わるネットワークになっている．

表 1　ノードの概要

CPU	AMD Barcelona, 2.3 GHz, 4 Cores
# Sockets	4
Memory	32 GB
Local Disk	SATA
Ethernet	Intel E1000 (1 Gbps)
	Myrinet 10G
OS	RHELS 5.1
File System	EXT3, NFS3

Catwalk 自体には計算ノード上にプロセスを生成する機能は含まれていないため，現在開発が進められている SCore[4] Version 7 に組込んで計測した．また結果の考察を単純化するために，本稿では計算ノードのそれぞれにひとつのユーザプロセスを生成するものとした．ファイルの I/O およびパイプラインを一度に転送するデータの大きさは全て 64 KB とした．NFS のマウントオプションおよび`/proc/sys/vm` 以下にあるバッファキャッシュに関するカーネルパラメータはデフォルトのままである．Ethernet の MTU はデフォルトの 1500 である．

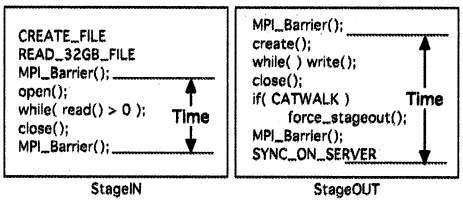

図 4　評価プログラムの疑似コード

図 4 に計測に用いたプログラムの疑似コードを示す．左側がステージイン，右側がステージアウトの計測プログラムである．ステージインの計測では，最初に読込むためのファイルを生成し，バッファキャッシュをフラッシュするためにメモリと同じ大きさの 32 GB のファイルを読み込む．バリア同期の後，全てのランク上で計測を開始する．同一のファイルを読込オープンし，ファイルを全て読み込んでからクローズするまでの時間をバリア同期の後で計測した．Catwalk の場合，`open()` が返ってきた時点で該当するファイルがステージインされている．

一方，ステージアウトの場合では，全てのランク上で異なるファイル名のファイルをオープンし，所定の量のデータを書き込む．通常の Catwalk の処理ではプログラムが終了してからステージアウトが行われるが，これだと計測の都合が悪い．そこで，強制的にファイルのステージアウトを起動し，全てのステージアウトが完了するまで待つという計測のための API を作り，ファイルをクローズした後にこの API 関数を呼ぶこととした．ファイルの最終的なディスクへの書込バンド幅を計測するために，この計測プログラムでの測定時間に，ファイルサーバ上での `sync` コマンドの実行時間を加えてある．

上記のプログラムで計測した時間から合計のバンド幅をプロットしたグラフが図 5 および図 6 である．図 5 は NFS，図 6 は Catwalk の結果である．比較が容易となるよう両方とも同じスケールとした．なお，ステージアウトではディスクの空き容量の制約から，最大 5 GB のファイルまで計測した．

NFS で並列にファイルを読み込んだ場合，ファイルサイズやノード数に無関係にほぼ 95 MB/s 前後の値を示した．通信ネットワークが 1 Gbps の Ethernet であるため，ネットワークのバンド幅がボトルネックになっていると推測される．一方，ファイルの並列書込では，合計のバンド幅は 20-40 MB/s と幅があり，ノード数が多い程，またファイルのサイズが大きい程，バンド幅は減少し 20 MB/s に収束する傾向が見られる．

Cawtwalk のステージインでは，おおまかな傾向として，1 計算ノードにつき 20 MB/s 弱のバンド幅が実現されていることが分かる．別な見方をすれば，Catwalk のステージインに要する時間は計算ノードの数よりもファイルの大きさの影響が大きい．このグラフからは，計算ノード数が 8 のときに NFS のバンド幅を超え，16 のときのバンド幅は 200 MB/s と NFS の倍以上の値を示している．この結果はパイプライン処理がほぼ理論通り実現されているものと考えられる．

NFS の場合はサーバ上のディスクから読込んだデータをネットワーク経由でクライアントに送り，ユーザプログラムのバッファに書込むだけなので，ファイルのアクセス回数は Catwalk よりも少ない．しかしながら NFS の 1 ノード時の読込バンド幅は 40 MB/s 程度と Catwalk の 2 倍程度しかない．

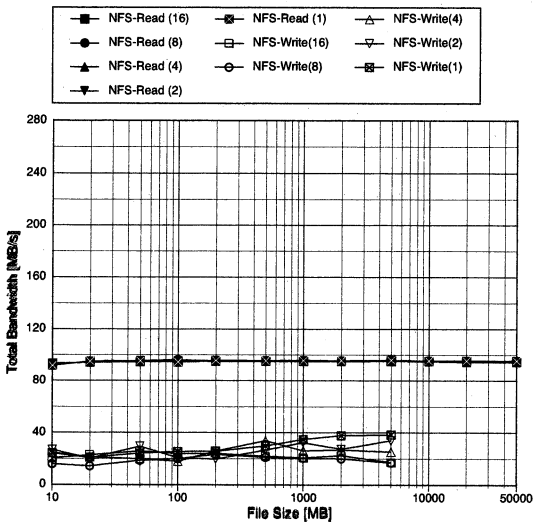

図 5　NFS の並列アクセス性能

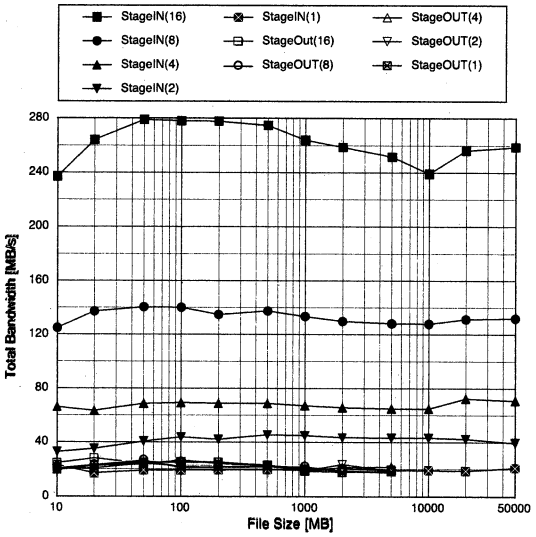

図 6　Catwalk のステージング性能

前述したように，ステージイン時のパイプラインがバッファキャッシュのフラッシュするストールするのを防ぐために，O_DIRECT モードでファイルを書き出すようにした．こうするとステージインが終わってからユーザプログラムが同じファイルを読込む際にバッファキャッシュの恩恵がない．別途，ローカルディスクに対し，Catwalk のステージインと同様のアクセス速度を計測した結果，ブロックサイズを 32 KB としたときの O_DIRECT モードでの書込速度は，およそ 40 MB/s であり，書込終了後，直ちに同じファイルを O_DIRECT を設定しない場合の読込速度は約 40 MB/s であった．Catwalk のステージインの速度が 1 ノードあたり約 20 MB/s なので，計算が合う．しかし 16 ノードでの最大バンド幅は 280 MB/s であり，320 MB/s には届いていない．この原因は，書込む対象となったファイルシステムが EXT3，つまりジャーナルファイルシステムであり，定期的にジャーナルファイルが書込まるためにパイプラインが乱れたものと推測される．O_DIRECT による同期的な I/O による性能低下も考えられることから，計算ノード上の書込みをマルチスレッド化するなどの非同期的な I/O にすれば改善される可能性があると考える．

一方，Catwalk のステージアウト処理はパイプライン処理が実現できないため，ステージアウト時間はファイルのトータルのサイズに比例する．図 6 から，Catwalk はおよそ 20 MB/s 程度のバンド幅を実現できている．ローカルディスクの速度が O_DIRECT の場合でも 40 MB/s あることを考慮すると，Catwalk の速度はまだ改善の余地が残されているものと思われる．Catwalk の 20 MB/s というバンド幅が計算ノード数が 8 以上の NFS とほぼ同じというのは興味深い．Catwalk ではローカルディスクからファイルを読み込み，転送し，サーバのディスクにコピーする．さらに Catwalk がステージアウトする以前に，ユーザのプログラムがそのファイルに書込んでいる．このように Catwalk のファイルのアクセス回数は NFS よりも多いため不利なはずである．NFS の場合では，計算ノードの数が多い場合にサーバにデータや処理が過度に集中し，性能が低下したものと考えられる．

3. 考　　察

オンデマンド化によりファイルのステージインはユーザプログラムの open() の実行時まで遅延される．プログラムの実行の前に，陽にステージインを開始しておくことで，プログラムで呼び出される open() 時のステージイン時間を減らすことが可能である．こうすることで，ステージイン処理とプログラムの実行の一部がオーバラップし，ジョブ全体の実行時間を短くする効果が期待される．このような動作はキャッシュのプリフェッチに近い．この時，事前のステージイン開始のファイル名が間違っていたとしても，正しいファイルが open() 時にステージインされるため，意図した動きではないが，プログラムは正しく実行される．

ステージングをオンデマンド化することは，ファイルのコピーを陽に記述しないことを意味する．これは遠隔にあるファイルをローカルディスクにキャッシュするのと同じとみなすことができる．例えば，ステージイン中にディスクが満杯になったため，既にステージインが完了したファイルを消したとしても，プログラムの実行は可能である．以上のようにステージングのオンデマンド化は，ステージング記述が不要になるというユーザの利便性以外にもメリットがあると考え

られる．

オンデマンド化を実現するにあたっては，LD_PRELOAD 環境変数による共有ライブラリのプリロード機能を用いているだけである．現在の glibc では，glibc の中から呼ばれた関数に対してフックを掛けることができない，という制約がある．このため Calwalk では，例えば fopen() 関数のフックも open() とは別に用意されている．C 言語以外の言語処理系では，実行時ライブラリから呼ばれる glibc の関数に直接フックを掛けることができない場合があり，言語処理系毎にフック関数を用意する必要がある．

本格的な並列ファイルシステムの多くは複数のファイルサーバと強力なネットワークを前提としているため，単一のファイルサーバと凡庸なネットワークという構成を基本とする Catwalk との直接的な比較は公平ではないと考える．一方，多くの並列ファイルシステムでは，巨大なファイルアクセスのバンド幅を重視して設計されており，大量の小さいファイルのアクセスに対して性能が発揮できないという問題を抱えていることが多い．Catwalk の実装からは，大量の小さいファイルへのアクセスにおいて大きく性能が劣化するとは考え難い．ファイルのアクセスパターンによっては，高価な並列ファイルシステムと対等の性能を発揮する可能性もある．また Catwalk と並列ファイルシステムは共存できるため，状況に応じて並列ファイルシステムと Catwalk を使い分けることも可能である．

本稿では，Catwalk の基本的な性能に注目し，評価をおこなってきた．様々なファイルアクセスパターンにおける性能，NFS や並列ファイルシステムと組み合わせた時の性能，実用化に向けたチューニングと性能評価を今後の課題としたい．

4. 関連研究

Catwalk と同様のパイプライン方式により，複数のローカルディスクにファイルをコピーする方式としては Dolly+[5] がある．論文 [3] ではリングトポロジー以外のトポロジーについて比較検討を行っている．ファイルコピーだけでなく，任意のコマンドに対して実行可能にしたものとして nettee[2] がある．また SCore[4] にも scout と言う名前の nettee に似た機能のコマンドがある．しかしながら，これらはファイルのブロードキャストやコマンドの並列実行が目的で，Catwalk のステージアウトに相当する機能はない．

PVFS[1] や Gfarm[6] などは，複数のファイルサーバによりバンド幅を稼ぐ設計になっており，単一のファイルサーバからはネットワークのバンド幅以上のファイルアクセスバンド幅を出すことができない設計になっている．Gfarm では読込んだファイルに変更を加えても反映されないという点で，Catwalk と似た側面を持っている．

おわりに

本稿では，オンデマンド化したクラスタ用ファイルステージングシステム Catwalk の概要とプロトタイプの設計実装ならびにその基本動作の評価結果を示した．Catwalk はファイルステージングをオンデマンド化し，ステージングの記述が不要になるという大きなメリットがある．また，Catwalk はユーザレベルで実現されていること，特別なハードウェアを必要としないことという 2 点から，ほとんどのクラスタ上で容易に試すことができる．

ファイルのステージインをリングトポロジーに沿ってパイプライン的にコピーすることで，計算ノード数にほぼ比例するバンド幅を実現することができた．1 台のファイルサーバにも関わらず．16 台の計算ノードの場合では，NFS（あるいはネットワークバンド幅）の 2 倍以上の性能を示した．一方，ステージアウトのバンド幅に関しては NFS と比肩できる性能を発揮することが示された．今後は Catwalk のチューニングとさらに詳細な性能評価を行うつもりである．Catwalk は SCore[4] の次バージョンに組込まれ，オープンソースとして配布される予定である．

謝　辞

本研究は文部科学省「e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」からの支援を受けている．

参考文献

1) Carns, P. H., Iii, W. B. L., Ross, R. B. and Thakur, R.: PVFS: A parallel file system for linux clusters, *In Proceedings of the 4th Annual Linux Showcase and Conference*, USENIX Association, pp. 317–327 (2000).
2) : nettee. http://saf.bio.caltech.edu/nettee.html.
3) Rauch, F., Kurmann, C. and Stricker, T.: Partition Cast - Modelling and Optimizing the Distribution of Large Data Sets in PC Clusters, *Euro-Par 2000 – Parallel Processing* (Bode, A. and Ludwig, T.(eds.)), Munich, Germany, Springer (2000).
4) : SCore. http://www.pccluster.org.
5) Takizawa, S., Takamiya, Y., Nakada, H. and Matsuoka, S.: A Scalable Multi-Replication Framework for Data Grid, *International Symposium on Applications and the Internet (SAINT 2005 Workshops)*, pp. 310–315.
6) 建部修見, 森田洋平, 松岡聡, 関口智嗣, 曽田哲之: ペタバイトスケールデータインテンシブコンピューティングのための Grid Datafarm アーキテクチャ, 情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム, No. Sig 6 (HPS 5), pp. 184–195 (2002).