Vol.2012-HPC-135 No.30
2012/8/2

並列ファイルシステムにおける設計トレードオフの検討

松葉 浩也[†]　　鵜飼 敏之[†]　　清水 正明[†]

HSFS はスーパーコンピュータにおいて大容量ファイルのファイル入出力を高速に行うことを目的に開発された並列ファイルシステムであり、大規模ファイルに対する入出力バンド幅において高い性能を有してきた。本稿はこの HSFS に対し、従来重視していなかったファイル作成削除などのメタデータ操作、あるいはバッファサイズの小さい入出力の性能を向上させる方法を検討したものであり、ブロックデバイスの抽象化方法およびキャッシュシステムについて性能トレードオフを議論する。検討の結果、特に従来アーキテクチャの主要コンポーネントであったサブファイルシステムアーキテクチャの一部を更新することにより、メタデータ操作における通信回数を削減し、ファイル作成性能を従来比で最大 30 倍に向上させることに成功した。

Design Tradeoffs in Parallel File Systems

HIROYA MATSUBA[†]　TOSHIYUKI UKAI[†]
MASAAKI SHIMIZU[†]

HSFS is a parallel file system product designed to provide large I/O bandwidth on large-scale I/O operations. This article discusses the design consideration for improving performance of HSFS, especially on that for metadata and small I/O operations. Two aspects of the software architecture are discussed, which are methods for abstracting block devices and cache system design. As a result of the discussion, we have re-designed HSFS and reduced the number of communication messages required for metadata operations. It is shown that performance on file creation has improved by 30 times, compared with that of the previous version.

1. はじめに

クラスタ型のスーパーコンピュータにおいては、クラスタを構成するすべてのノードに共通のファイルシステムイメージを提供するために共有ファイルシステムを用いる場合がある。特に規模の大きなスーパーコンピュータにおいては、多数の計算ノードからの入出力要求に耐え、計算性能に見合った I/O 性能を提供するために、複数のサーバに負荷を分散しながら共有ファイルシステムの機能を提供する並列ファイルシステムを用いることが多い。

HSFS (Hitachi Striping File System) は（株）日立製作所が製造、販売する並列ファイルシステムであり、公共機関のスーパーコンピュータや企業情報システム等での稼動実績がある。HSFS は I/O バンド幅において高性能ストレージの性能を最大限に引き出すことを目標に設計されてきたファイルシステムである。2002 年まではスーパーコンピュータ製品である SR8000 のファイルシステムとして利用され、ハードウェア性能の 90%以上の実効性能を提供していた。その後、HSFS はスーパーテクニカルサーバ SR11000 のための並列ファイルシステムとして AIX オペレーティングシステムに移植され、2005 年には 128 ノード構成の SR11000 システムにおける実測の並列入出力性能として 20GB/s を記録している。

HSFS はバッチ運用のスーパーコンピュータにて、大量の入出力を行うスーパーコンピュータアプリケーションで高い性能が得られるよう最適化してきている。利用者がファイルシステムのレスポンスを感じる場面の少ないバッチ環境では、対話的操作の際の利用感に関わるファイル作成、削除などのメタデータ操作性能、あるいは数キロバイトの比較的小さなバッファサイズで発行される入出力命令（以下「小規模 I/O」と呼ぶ）の処理性能は重要ではなく、HSFS ではそれらを重要度の低い性能指標としてきた。

ところがスーパーコンピュータを取り巻く環境は変化し、2008 年のオープンスーパーコンピュータ[1]に代表されるようなコモディティーサーバを用いた多ノードクラスタが一般的となった。このようなスーパーコンピュータにおいては、全ノードがバッチ運用されるとは限らず、また、小規模 I/O を発行するオープンソースソフトウェアも利用される。このような新しいタイプの利用形態において HSFS はオーバヘッドの目立つファイルシステムとなってきており、性能問題を指摘する文献も発表されている[2][3]。

このような状況を改善するため、我々は 2010 年より HSFS の基本アーキテクチャを全面的に見直し、メタデータ操作や小規模 I/O においても一定の性能が得られる新たしいソフトウェアを設計、開発した。本稿はこのアーキテクチャ刷新にあたって検討した設計上のトレードオフについて述べるものである。具体的には「サブファイルシステムアーキテクチャ」と呼ばれる基本設計、およびキャッシュ層の設計について述べる。前者については HSFS が内部

[†] (株)日立製作所 中央研究所
Central Research Laboratory, Hitachi Ltd.

的に利用していたネットワークファイルシステムの利用をとりやめて独自の簡潔なプロトコルを定義することで大幅な通信回数削減を達成したことを述べる。後者についてはHSFS のファイル分割処理に依存しないよう、分割処理より上位のソフトウェア層にキャッシュ機能を設け、小規模I/O をキャッシュで高速化することを可能にしたことを述べる。

2. HSFS の構成

本章ではアーキテクチャ議論に先立ち、新旧 HSFS に共通する基本的な構成を述べる。

2.1 基本構成

図 1 に HSFS の基本構成を示す。HSFS は単一のメタデータサーバと複数のデータサーバを用いて単一のファイルシステムツリーを提供する分散・並列ファイルシステムである。

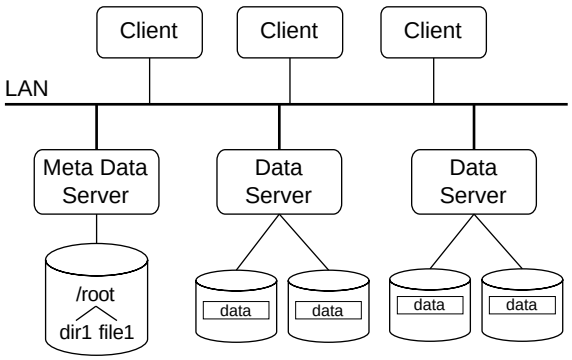

図 1　HSFS の基本構成

メタデータサーバは単一のディスクボリューム（物理ディスク、パーティション、LVM の論理ディスク、ソフトウェア RAID など任意のブロックデバイス、以下それらの総称として「ディスクボリューム」と表記する）を配下に持ち、ディレクトリツリーとファイル属性を保持する。ディレクトリツリーはディスクボリューム上に構成された UFS ファイルシステムによって管理されている。HSFS 上のディレクトリはこの UFS 上のディレクトリと一対一で対応しており、ディレクトリに関してはサーバ上のディレクトリ名や属性がそのままクライアントにも公開される NFS と同等の構成である。シンボリックリンク、ハードリンク、デバイスファイルなどの特殊ファイルもディレクトリ同様、メタデータサーバ上の UFS に実体を作成し、各クライアントからのアクセスを許可する構成である。

データを格納する通常ファイルに関しては、HSFS は NFS とは異なり、メタデータサーバ上の UFS には管理情報のみを記録したファイルを作成する。この管理情報にはデータ本体を保持するデータサーバの位置等が書かれている。ファイル内容にアクセスするクライアントはメタデータサーバからこの管理情報を取り寄せ、この情報を手がかりにデータサーバと直接通信することによりファイルに格納されたデータにアクセスする。

このように HSFS はブロックデバイスであるディスクを抽象化してファイルを格納する機能（Linux における ext4 に相当）およびネットワークを用いた通信を行い遠隔のサーバに格納されたファイルへのアクセスを提供する機能を併せ持つ。本稿では前者の機能を「ディスクファイルシステム」、後者の機能を「ネットワークファイルシステム」と呼ぶ。

なお、HSFS は各サーバに待機系を設定し、故障時に待機系に動作を移すフェイルオーバー機能を有するが、本稿では簡単のため、特に必要な場合以外は図示も言及も行わない。また、HSFS はストレージエリアネットワークを経由して共有したディスクを複数のサーバからアクセスすることを可能とする「SAN 共有機能」も有する。スパコン以外ではこの機能が用いられることが多く、同じ製品の機能であるが、SAN 共有機能は本稿で述べる技術とは独立であり、想定する製品の適用範囲や性能特性が異なる。SAN 共有時の動作については本稿の議論は当てはまらないので注意が必要である。

2.2 実装

HSFS は AIX または Linux で動作する並列ファイルシステムである。いずれの OS においてもカーネルエクステンション(AIX)あるいはカーネルモジュール(Linux)としてオペレーティングシステムのカーネルに直接組み込むソフトウェアとして実装されている。AIX、Linux いずれの場合もオペレーティングシステムの VFS(Virtual File System)層に接続しており、OS 標準のマウントコマンドを利用して UNIX ファイルシステムツリーの一部として利用可能なファイルシステムである。FUSE[4]のようなユーザー空間ファイルシステムライブラリや、HSFS 専用の入出力ライブラリは不要であり、アプリケーションからは標準的な POSIX インタフェースで操作可能である。

HSFS は日立製スーパーコンピュータ SR8000 で用いられていたオペレーティングシステムのカーネルおよび分散ファイルシステム部分を AIX に移植し、さらに並列ファイルシステム機構を加えて誕生した経緯がある[5]。今日では SR8000 由来の部分と Linux を接続するレイヤーも実装されており、HSFS は AIX、Linux 両 OS で動作する。

2.3 管理系

ファイルシステムとしての本質的な入出力機能に加え、HSFS はクラスタの死活監視の機能を持つ。この機能が活躍するのは例えばネットワークに障害が発生し HSFS クラスタが分断された場合である。分断によって生まれた複数

のサブクラスタが、それぞれにフェイルオーバー処理などを行い動作を継続してしまうとファイルシステムを破壊してしまうため、このような場合には死活監視機能が過半数を維持したクラスタのみに動作を継続させる。これはいわゆるスプリットブレインシンドロームを防止する仕組みであり、広く知られたものであるため本稿では詳細は述べない。

なお、HSFS がフェイルオーバー機能を有さなかった時代の死活監視機能は、サーバのダウンを検出した場合に全ノードを停止していたが、現在では過半数の I/O サーバが同時に停止するような極端な場合を除いては全系停止が発生することがない設計に改められている。

3. HSFS 新アーキテクチャの検討

本章では HSFS の新しいアーキテクチャについて、設計の過程で検討したトレードオフと共に議論する。一章で述べたように、新アーキテクチャ設計の目標はメタデータ操作と小規模 I/O の高速化である。本章ではサブファイルシステムアーキテクチャおよびキャッシュ層の設計について検討するが、前者が主にメタデータ操作性能、後者が主に小規模 I/O 性能に関わる設計ポイントである。

3.1 サブファイルシステムアーキテクチャの検討

以前のバージョンの HSFS はシングルサーバのネットワークファイルシステム（一般的な NFS と同等の動作）とディスクファイルシステムの組み合わせを「サブファイルシステム」として利用し、このサブファイルシステムを利用してファイルの並列処理を実装していた。これを「サブファイルシステムアーキテクチャ」と呼んでいる。本節ではこのアーキテクチャの利害得失について検討する。

3.1.1 サブファイルシステムアーキテクチャの概要

図 2 にサブファイルシステムアーキテクチャを採用した従来 HSFS のソフトウェアスタックを示す。図に示すように並列ファイルシステムの主要コンポーネントであるクライアント側のストライピング処理は、OS の VFS からの命令を直接受ける位置に配置されており、ストライピング処理層は下位に位置するサブファイルシステム（図中でネットワーク FS、ディスク FS と表記）を利用してディスクへのアクセスを行っている。

動作原理を図 3 に示す。図中，実線の楕円で示したファイルシステムはディスクを伴ったファイルシステムである。点線の楕円で示したファイルシステムは，ディスクを伴う遠隔のファイルシステムをネットワークファイルシステム経由でマウントし，仮想的にローカルファイルとしてアクセス可能としたものである。また、マスターファイルシステム（図中、マスターFS と表記）とはメタデータサーバが保持するディレクトリ構造やファイル属性を記録するファイルシステムである。サブファイルシステム（図中、サブ FS と表記）とはデータサーバが保持するファイル内容を記録するファイルシステムである。

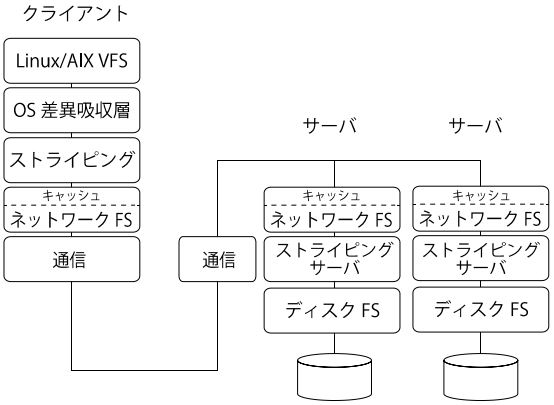

図 2 サブファイルシステムアーキテクチャ

図に示すよう，以前のバージョンの HSFS はネットワークファイルシステムを用いてマスターファイルシステムおよび複数のサブファイルシステムをすべてクライアント計算機にマウントしている。クライアントプログラムは仮想的にローカルファイルとしてアクセス可能となったネットワークファイルシステムのファイルに対して，ファイル作成，書き込みなどの操作命令を発行することで，間接的にファイルサーバ上のデータを更新している。

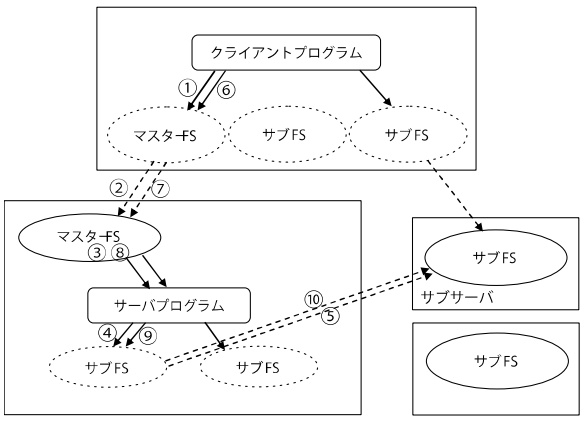

図 3　サブファイルシステムの動作

メタデータサーバがサブファイルシステムにアクセスする際も同様であり，メタデータサーバのプログラムは，マスターサーバにマウントしたサブファイルシステムに対してファイル操作命令を発行している。例えばファイルの作成処理における動作を示すと次のようになる。

① ファイル作成のシステムコールを受けたクライアントプログラムは、クライアントにマウントされているマスターファイルシステムに対してファイル作成命令を発行する
② マスターファイルシステムを管理するネットワークファイルシステムを通じてメタデータサーバにファイル作成命令が伝えられる
③ メタデータサーバにてディスク上にファイルが作成される
④ マスターサーバがマスターサーバにマウントされているサブファイルシステムに対してサブファイル作成命令を発行する
⑤ ブファイルシステムを管理するネットワークファイルシステムを通じてファイル作成命令がサブサーバに伝達されファイルが作成される

以下，⑥より⑩まで，ファイルの属性設定について上記①から⑤と同様の流れで、新規ファイルの属性が設定される。

サブファイルシステムアーキテクチャには次の２つの側面がある。

1. ディスク抽象化機能の分離 : 並列ファイルシステム上のデータやメタデータをディスクファイルシステム上のファイルとして保持することで、並列ファイルシステムからディスクのブロック管理機能を分離する。
2. 通信機能の分離 : ネットワークファイルシステムに通信機能を持たせることで、並列ファイルシステムから通信機能を分離する。

3.1.2 利害得失の検討

サブファイルシステムアーキテクチャによるディスク抽象化により、ディスクへのアクセスはすべてディスクファイルシステムを経由することになる。ディスクファイルシステムは各ノード、ディスクドライブ（通常は RAID の論理ユニット）ごとに独立して動作が可能である。また、ディスクファイルシステムにおいてはブロック割り当てや先読み、まとめ書きなどの最適化手法が確立しており、アーキテクチャ刷新以前より HSFS が大規模 I/O で高い性能が得られたのは、これらの最適化が行われてきたためである。ディスク抽象化機能としてのサブファイルシステムは、サブファイルシステムを用いず分散ファイルシステムがクラスタ全体のディスクについてブロックレベルの管理を行う場合に比べて、並列性や既存技術の利用可能性が高いという利点が認められる。

一方、サブファイルシステムアーキテクチャによる通信の抽象化は、並列ファイルシステムのデータ分散処理をローカルファイルシステムへの操作と同等に記述することを可能とし、ソフトウェア構成の単純化をもたらす。しかし、

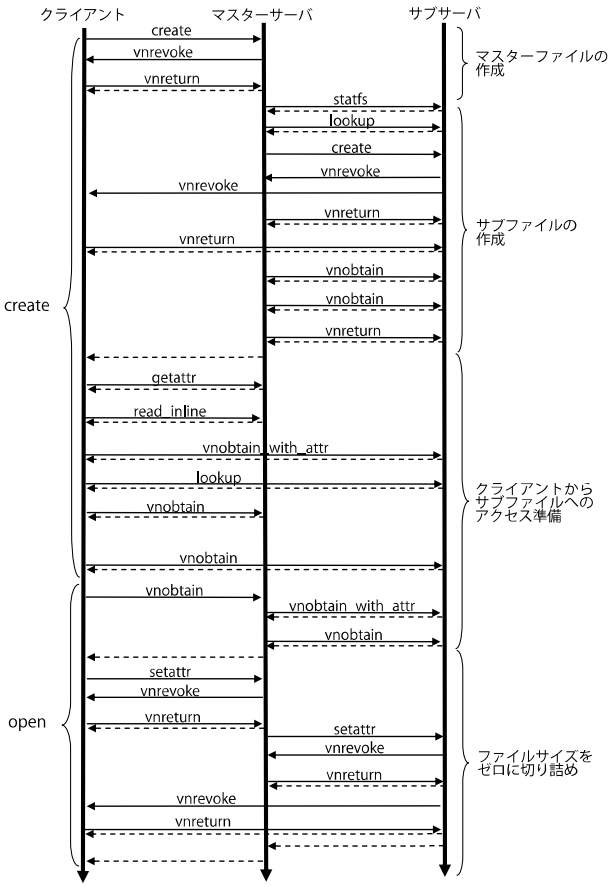

図 4　従来のファイル作成時における通信

HSFS においては副作用もあり、サブファイルシステムが多くの通信を発生させる場合があった。図 4 は顕著な例であり、従来 HSFS でファイルを作成した場合に発生していた通信のトレース結果である。単一のファイルの作成のために 53 回の通信が発生しており、HSFS のメタデータ操作のオーバヘッドが大きい要因であった。これはサブファイルシステムとして用いているネットワークファイルシステムのキャッシュ一貫性制御の影響である。HSFS が、性能よりも一貫性の厳密な保持を優先したネットワークファイルシステムを用いていたこと、およびファイル作成の一連の操作の中で、並列ファイルシステム層がクライアント、サーバの双方で同じファイルにアクセスすることにより、キャッシュのフラッシュに関わる通信が必ず発生してしまうことなどが、通信回数を増加させる原因となっていた。

3.1.3 新 HSFS における判断

前節までに議論したよう、HSFS におけるサブファイルシステムアーキテクチャについては、ディスクの抽象化には利点が認められるものの、ネットワーク抽象化についてはデメリットが目立つ状況であった。そのため、新アーキテクチャの策定にあたっては、ディスクの抽象化にディスクファイルシステムを用いる設計は踏襲しつつ、通信は並列ファイルシステム自らが管理し、通信の回数を最小限に抑える設計とした。

HSFS に限らない一般論としても、並列ファイルシステムを実装するために必要な通信インタフェースを一般的なネットワークファイルシステムのインタフェースに置き換えてしまう（例えば、NFS プロトコルのみを用いて並列ファイルシステムを記述する）ことは、並列ファイルシステム層の自由な通信を阻害することにつながり、通信回数を最小化することの妨げになると考えられる。

3.2 キャッシュ層の検討

一般的にネットワークによる一定の遅延が避けられない並列ファイルシステムにおいては、クライアントからのシステムコールが発行されるたびに通信を発生させるのでなく、利用頻度の高い情報をクライアント側にキャッシュすることは高速化のために必須である。

すでに図 2 に示したように、従来の HSFS はサブファイルシステムアーキテクチャとなっており、キャッシュ機構はストライピング層の下位に位置するネットワークファイルシステムが提供していた。前節で検討したように新アーキテクチャではネットワークファイルシステムの利用は取りやめることとしたため、新たにキャッシュを導入する必要がある。

キャッシュを導入する層としては、ストライピング処理層の下位に配置する方法と上位に配置する方法の 2 通りが考えられる。ストライピング処理の下位に配置した場合は、メタデータを格納するファイルシステムに関してもデータを格納するファイルシステムに対しても同じキャッシュが有効となる。しかし、本来、メタデータとデータでは必要となるキャッシュポリシーは異なるため、それらを一律に扱うのは無駄な一貫性制御の原因となる。また、キャッシュ層をストライピング処理の下位に配置する別の問題として、ストライピングの方針を変更すると最適なキャッシュポリシーまで変わってしまう可能性も考えられる。これらの問題を回避するため、新アーキテクチャにおいてはキャッシュ層をストライピング層の上位に配置した。

この設計により、以前の HSFS が分割したサブファイル単位でキャッシュ制御を行っていたのに対し、新アーキテクチャではユーザーに見えるファイルの単位でキャッシュ制御を行うこととなった。新しいキャッシュ層の実装にあたっては従来の HSFS で問題となっていた小サイズ I/O 性能問題を解決すべく、書き込み時のまとめ書き、読み込み時の先読み等、小サイズ I/O のオーバヘッドを吸収する仕組みを実装している。

3.3 HSFS 新アーキテクチャ

本章では主にメタデータ操作高速化のためにサブファイルシステムアーキテクチャの見直し、小規模 I/O の高速化のためにキャッシュ層の見直しを行った。この二点ですべての要素を議論し尽くせているわけではないが、前節までの議論等を踏まえ策定した新 HSFS のアーキテクチャ概要を図 5 に示す。

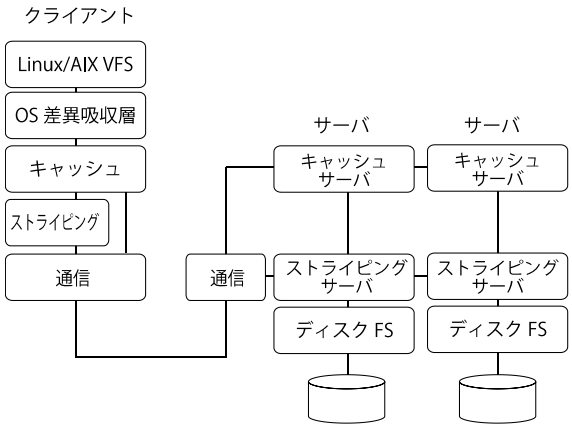

図 5　新 HSFS のアーキテクチャ

新しいアーキテクチャではサブファイルシステムはディスク管理部分のみで利用しており、従来アーキテクチャのネットワークファイルシステムに相当する仕組みは存在しない。代わりに新しく作成するストライピング層が最小限の通信回数でサーバ側と通信する。キャッシュはストライピング層の上位に位置し、ストライピング層の影響は受けない。将来ストライピング層にアーキテクチャ変更があった際もキャッシュは影響を受けないよう、サーバ側のキャッシュ管理機構との通信はキャッシュ層自らが通信層を利用する設計としている。

従来、53 回の通信が必要であったファイル作成は、新アーキテクチャではキャッシュの状態にも依るが図 6 に示すように最大でも 6 回となっている。

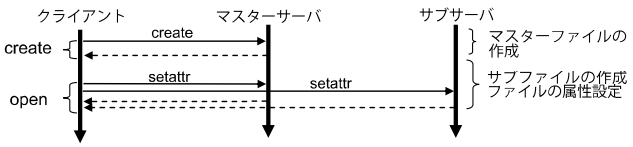

図 6　新アーキテクチャにおけるファイル作成

4. 性能

本章では新アーキテクチャに基づき実装した新しい HSFS の性能について述べる。性能評価は 32 台のクライアント、8 台の I/O サーバ、16 台のストレージを InfiniBand QDR で接続したクラスタを用いて行った。I/O サーバのうち 1 台はメタデータサーバとデータサーバを兼ねており、残りの 7 台はデータ専用のサーバである。表 1 に各マシンの仕様を示す。

表 1 評価環境（1 台あたりの仕様）

構成要素	I/O ノード	クライアント
CPU	Xeon 2.93GHz x 12	Opteron 2.6GHz x 8
Memory	96GB	32GB
Network	InfiniBand QDR	InfiniBand QDR
Storage I/F	8Gbps FC x 2	
Storage	Hitachi AMS2300 x 2	

4.1 メタデータ操作性能

メタデータ操作の性能を測定するために mdtest[6]を用いたベンチマークを行った。本稿で述べたメタデータ操作性能改善策は主にスループットでなくレイテンシの削減を目指したものであるため、本測定でもレイテンシが観測できるよう 1 ノード 1 プロセスでの実行とした。図 7 に結果を示す。

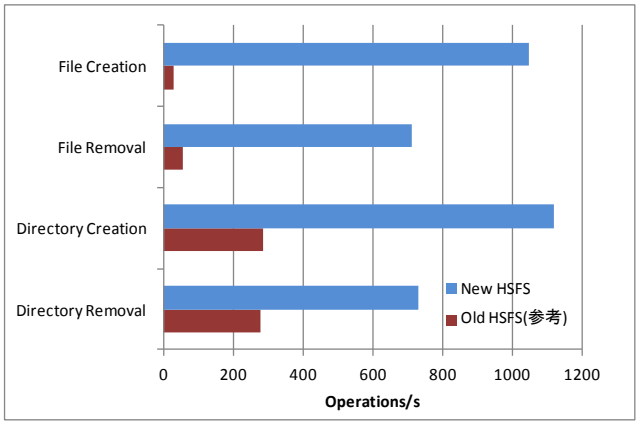

図 7　メタデータ操作性能

“New HSFS”のグラフが本稿で述べた新アーキテクチャの HSFS における結果である。参考データとしてバージョン 4 と呼ばれていた旧アーキテクチャでのデータを”Old HSFS”として示す。これは I/O サーバに POWER プロセッサを搭載した計算機を用いた場合の測定結果で、ベンチマークソフトウェアも mdtest でなく自作のテストプログラムであるため平等な比較ではないが、テスト内容は両者に本質的な差はなく、プロセッサネックの処理が含まれるベンチマークでもないため、一定の意味は持つと考えられる。

測定の結果、ファイル作成・削除、ディレクトリ作成・削除で新バージョンは旧バージョンのそれぞれ 30 倍、10 倍、4 倍、3 倍の性能となっており、新アーキテクチャで高速化に成功したことがわかる。

4.2 小サイズ I/O 性能

小サイズ I/O の改善を確認するために 1 クライアント上の単一プロセスから HSFS 上のファイルに書き込みを行い、その際のシステムコールのバッファサイズを変化させて I/O 性能を測定した。図 8 に結果を示す。参考までに旧バージョンで行った同等の測定結果と併せて示すが、前節と同様ハードウェアとベンチマークソフトが異なるため単純な比較はできない。特に旧バージョンの測定結果は、単一の内蔵ハードディスクしか持たないサーバでの結果であるため、ソフトウェアにまったくオーバヘッドがない場合でも 50MB/s 程度の性能しか期待できない。

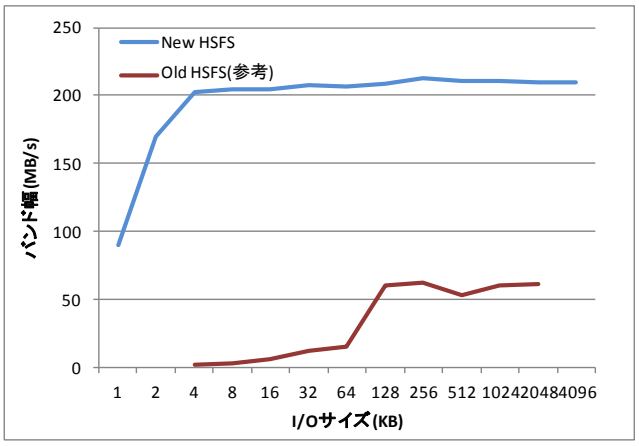

図 8　小サイズ I/O バンド幅

上に述べたようにハードウェア環境の差により期待される最大バンド幅は異なるが、4KB から 64KB のバッファサイズにおいて、旧バージョンでは内蔵ハードディスクの性能も下回っているのに対し、新バージョンではほぼディスク性能が得られていることがわかる。

4.3 全体 I/O バンド幅

クラスタ全体で得られる I/O バンド幅を測定するために IOR[7]を用いた性能測定を行った。本測定では IOR を各プロセスが独立したファイルを書き込むモードに設定し、実行プロセス数を 1 プロセスから 64 プロセスまで変化させて総合バンド幅を測定した。図 9 が測定結果であり、高いスケーラビリティが得られていることがわかる。なお、ストレージ自体の性能は RAID 構成やハードディスクの単体性能、搭載数、搭載場所により変化するため単純には議論できないが、図 9 に示した性能は予備評価として同一構成で測定した RAW デバイスとしてのストレージ性能のほぼ 100%を達成していることも確認できている。

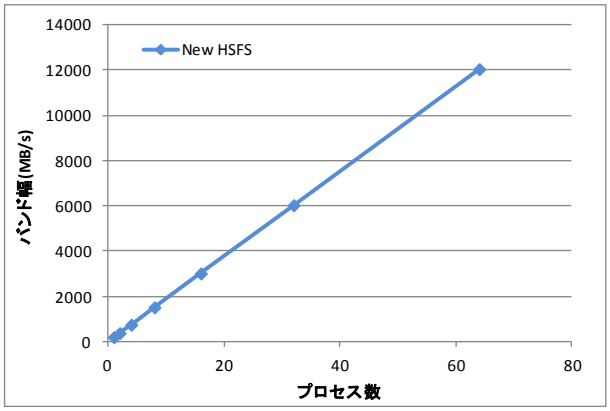

図 9　総合 I/O バンド幅

5. おわりに

本稿では近年アーキテクチャを刷新した並列ファイルシステム HSFS について、そのアーキテクチャ変更のポイントを述べた。具体的にはサブファイルシステムアーキテクチャおよびキャッシュシステムについて検討し、特に多数の通信を発生させていたネットワークサブファイルシステム部分について、その使用を取りやめ、簡素なアーキテクチャに変更した。また、新しい HSFS は従来のサブファイル単位のキャッシュでなく、ユーザーから認識可能な実ファイルの単位でキャッシュ制御を行う仕組みに変更し、I/O サイズに関わらず一定の性能を得るためのキャッシュシステムを実装している。これらのアーキテクチャ刷新によりファイル作成時の通信回数削減に成功し、性能は最大 30 倍に向上した。また新たに実装したキャッシュ層の働きにより小サイズ I/O の性能が改善し、書き込みシステムコールのバッファサイズが 4KB 以上であればほぼ一定の性能が得られることが確認できた。

本稿は日立製作所で設計、開発している HSFS の新旧バージョンを比較する形で議論を行ったが、設計トレードオフの議論は HSFS 特有のものでなく並列ファイルシステムの設計一般に応用できるものである。性能評価については新旧 HSFS の性能を示すにとどまっているが、他の並列ファイルシステムを HSFS と同一の環境に準備し、双方を十分にチューニングした上で公平な性能測定を行い、HSFS の客観的な実力を明らかにすることは今後の課題である。

参考文献

1 オープンスーパーコンピュータ http://www.open-supercomputer.org/
2 田浦健次朗: HA8000 のファイルシステムについて，スーパーコンピューティングニュース（東京大学情報基盤センター）, Vol.12 No.2
3 坂本雄三: 市街地における風・温熱・光・音環境総合数値予測データベースの開発，東京大学情報基盤センター平成 21 年度公募型プロジェクト報告会「ペタ/エクサスケールコンピューティングへの道 2010」
4 FUSE: Filesystem in Userspace http://fuse.sourceforge.net/
5 清水正明，戸部和政，人見洋一，鵜飼敏之，三瓶英智，飯田恒雄，藤田不二男: クラスタ環境におけるシングルシステム機能の実現，先進的計算基盤システムシンポジウム SACSIS 2006
6 mdtest HPC Benchmark, http://sourceforge.net/projects/mdtest/
7 IOR HPC Benchmark, http://sourceforge.net/projects/ior-sio/