Vol.2018-HPC-165 No.7
2018/7/30

計算ノードBurst Buffer実現のための データ処理方式の検討と評価

岡本拓也[†1] 酒井憲一郎[†1] 住元真司[†1] 石川裕[†2]

概要：Burst Bufferを計算ノードで実現するためには，限られたリソースで高いI/O性能を引き出し，かつ，I/O処理が同計算ノード上で実行されるアプリケーションに与える影響を低く抑える必要がある．本論文では，Burst Bufferのデータ処理に焦点を当て，検討と評価を行った．前者の課題を改善するために，固定長のバッファを用いたRDMAとDMAによるデータ処理を行った．また，後者については，システムタスクおよび割込み処理を計算コアから分離することで，計算に与えるノイズの低減を実施した．評価の結果，1コアでデバイス性能を出し切れることが分かり，また，10万並列規模でもアプリケーションの性能劣化が5%に抑えられることを確認した．

A Data Processing Method for Compute Node Burst Buffer

TAKUYA OKAMOTO[†1] KENICHIRO SAKAI[†1] SHINJI SUMIMOTO[†1]
YUTAKA ISHIKAWA[†2]

1. はじめに

スーパーコンピュータの演算能力向上に伴い，ストレージシステムの更なる高性能化,大容量化が求められている．従来のスーパーコンピュータは，計算ノードクラスタとHDDベースの共有ファイルシステムによって構成されることが一般的であった．しかし，更なる高性能化，大容量化に対応するためには, HDDベースのファイルシステムを拡張するだけでは難しい．そのため，NVMe SSDのような半導体デバイスを搭載した高性能レイヤとHDDベースの大容量レイヤに階層化することで，高性能かつ大容量なストレージシステムの実現を図ることが主流となっている．

Burst Bufferは，階層化ストレージにおける高性能レイヤとして研究開発が進んでいるシステムである．2018年6月にTop 500で世界1位を獲得したSummit[1]を始め，多くのスーパーコンピュータに導入されている[2][3][4][5]．Burst Bufferを導入すると，ファイルデータがBurst Buffer上にバッファリング，またはキャッシングされるため，アプリケーションのI/O時間を短縮できる．

Burst Bufferは，計算ノードとは別の専用サーバ（以降，専用サーバBurst Bufferとする）によって構成されるもの[6][7]とSSDを搭載した計算ノードによって構成されるもの（以降，計算ノードBurst Bufferとする）[8][9][10]が存在する．現在我々が開発しているLightweight Layered IO-Accelerator（LLIO）は，近年のスーパーコンピュータシステムにおいて電力制約が厳しくなっている背景を踏まえ，新規ハードウェアの追加が少ない計算ノードBurst Bufferの実現を目指し開発している．

計算ノードBurst Bufferでは，アプリケーションが実行される計算ノード上でBurst Bufferの処理を行う．そのため，限られたCPU，メモリリソースで高いI/O性能を引き出し，かつBurst Bufferの処理がアプリケーションの外乱とならないようにする必要がある．計算ノードBurst Bufferに関する研究[8][9]では，これらの課題に関する検討や評価は行われていない．

そこで本論文では, これらの課題を解決するためのLLIOのデータ処理について述べる．LLIOでは，計算ノード間および計算ノードと共有ファイルシステム間のデータ転送，SSDへのREAD/WRITEは, 固定長バッファを用いたRDMA，およびDMAで行う. これにより, データ処理に要するCPU負荷を低減すると共に，I/O性能のオーバーヘッドとなるページキャッシュの獲得・解放や，メモリコピーの発生を抑え，省資源型のデータ処理が可能となる．また，Burst Bufferの処理をはじめ，システムタスクや割込みを処理するコアと計算を行うコアを分離することで，I/O処理によるノイズを削減し，計算時間のばらつきを抑える．本論文では，これらのデータ処理方法の有効性を検証するための評価を行い，1コアに満たないCPU消費率でSSDのデバイス性能が引き出せること，I/O処理中の計算時間のばらつきが5%未満に抑えられることを示す．

2. 背景

本章では，Burst Bufferの分類と，計算ノードBurst Buffer実現に向けた課題について述べる．

2.1 Burst Bufferの分類

Burst Bufferとして研究開発されているシステム[6][7][8][9][10][11]は, ハードウェア構成, ユーザビューの観点でそれぞれを分類できる．

†1 富士通株式会社
FUJITSU LIMITED
†2 理化学研究所 計算科学研究センター
RIKEN Center for Computational Science

ⓒ2018 Information Processing Society of Japan

(1) **ハードウェア構成**

- **専用サーバ** Burst Buffer

  計算ノードとは別に，Burst Buffer 機能を持つ専用サーバを用いる方式である．ネットワークや CPU，メモリなど専用のハードウェアを追加する必要があるが，Burst Buffer の機能がリソースを占有できるため，高性能化しやすい．

- **計算ノード** Burst Buffer

  一部，または全ての計算ノードに SSD を搭載し，計算ノードの CPU，メモリなどのリソースを使用して，Burst Buffer の機能を実現する方式である．スーパーコンピュータでは，計算を行うアプリケーションはバッチジョブとして実行されることが一般的であり，物理的にまとまった単位で計算ノードがジョブに割り当てられることが多い．この時，ジョブに割り当てられた計算ノード群で Burst Buffer を構成することでジョブ間の干渉を抑えたファイルアクセスを実現できる．

(2) **ユーザビュー**

- **ノード内ローカル領域**

  各計算ノードで独立した名前空間を持つ．ノード間でファイル共有は出来ないが，名前空間やデータの管理がノード間で独立しているため，ノード数に比例して性能をスケールさせやすい．ノード内ローカル領域と共有ファイルシステム間のデータ転送は，ユーザが明示的にファイルを指定する．

- **ジョブ内ローカル領域**

  ジョブ間で独立した名前空間を持つ．ノード内ローカル領域とは異なり，ノード間でのファイル共有が行える．ノード内ローカル領域と同様に，ジョブ内ローカル領域と共有ファイルシステム間のデータ転送は，ユーザが明示的にファイルを指定する．

- **キャッシュ領域(共有ファイルシステム透過)**

  共有ファイルシステムと同じ名前空間を持つ．そのため，アプリケーションは共有ファイルシステムに透過的にアクセスが可能である．キャッシュ領域と共有ファイルシステム間のデータ転送はシステムが自動的に行う．

既存の製品や研究，そして現在我々が開発を進めている Lightweight Layered IO-Accelerator（LLIO）を上記の観点で分類したものを表 1 に示す．LLIO は，近年スーパーコンピュータにおいて電力制約が厳しくなっている背景を踏まえ，新たなハードウェアの追加が少ない計算ノード Burst Buffer の実現を目指して開発を進めている．また，3 種類の名前空間は，一時ファイルはノード内ローカル領域，ジョブ内ローカル領域に配置し，保存ファイルのみキャッシュ領域に配置する，といったように，ファイル種別によって使い分けることでアプリケーションの I/O 性能をより向上させることができるため，上述の 3 つのユーザビューを提供する．

表 1 Burst Buffer の分類

		ハードウェア構成	
		専用サーバ Burst Buffer	計算ノード Burst Buffer
名前空間	ノード内ローカル	DataWarp[7]	LLIO
	ジョブ内ローカル	DataWarp[7]	BeeOND[11], BurstFS[8], FusionFS[9] / LLIO
	キャッシュ	IME[6]	LLIO

2.2 計算ノード Burst Buffer 実現に向けた課題

計算ノード Burst Buffer では，アプリケーションが実行される計算ノード上で Burst Buffer の処理を行うため，以下の二つの課題が存在する．

(1) **省資源型高速データ処理の実現**

Top500 にランクインしている全てのシステムで用いられている Linux では，ファイルアクセスを高速化するために，すべてのファイルシステムで共通利用するページキャッシュ機構を導入している．カーネル内のすべてのファイルアクセスはこのページキャッシュを介して行われるため，データのファイル入出力の際にページキャッシュとデータ領域間でのコピーが必要になる．このため，ページキャッシュにデータをコピーするための CPU リソース，ページキャッシュのためのメモリリソースを必要とする．

しかし，計算ノード上のリソースは，本来アプリケーションのためのものである．そのため，Burst Buffer の処理が消費する CPU，メモリリソース量を小さく抑えながらも，I/O 性能を引き出す方法を検討する必要がある．アプリケーションの中には，I/O マスタとなるプロセスが計算に必要となるデータの READ，計算結果の WRITE を一括して行う，といったアプリケーションも存在する．そのため，プロセス多重度を上げた場合のスループットだけでなく，単一プロセスからの I/O 性能も高める必要がある．

(2) **I/O 処理ノイズの削減**

並列アプリケーションでは，実行中に同期を行うものが多い．同期点では，同期を行う全プロセスの計算完了を待つ必要があるため，最も計算の遅いプロセスが計算を終えるまで遅延される．スーパーコンピュータ上で実行されるアプリケーションには並列度が 10 万規模に及ぶものもあり，計算時間のばらつきは大きな性能劣化に繋がる．計算時間のばらつきの主たる原因として，計算を行っているコアに対するデバイスからの割込みや OS によるスケジューリングがある．そのため，I/O 処理によって計算コアへの割込みやシステムタスクのスケジューリングが発生しない

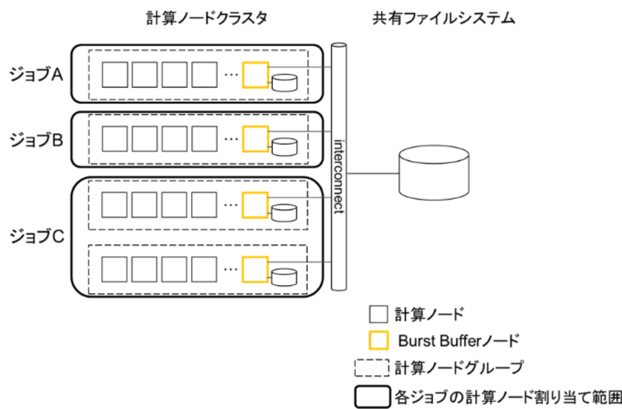

図 1 LLIO のシステム構成

ようにする必要がある．

3. LLIO の概要と実装

本章では，2.2 で挙げた課題に対する，LLIO の設計について述べる．

3.1 LLIO の概要

3.1.1 システム構成

図 1 に LLIO のシステム構成を示す．LLIO は，小規模な環境からエクサスケールのスーパーコンピュータなど非常に大規模なシステムでの運用を想定している．大規模なスーパーコンピュータでは，計算ノード数が十万ノード規模と非常に多くなり，その全てに SSD を搭載することは，コスト的に困難であると考えられる．そのため LLIO では，一部の計算ノードに SSD を搭載し，SSD が搭載されていない計算ノードは，SSD を搭載した計算ノード（以降，Burst Buffer ノードとする）にリクエストを発行することで，Burst Buffer の機能を利用するサーバ・クライアント方式を採用する．計算ノードと Burst Buffer ノード，共有ファイルシステム間は，InfiniBand など RDMA 通信をサポートするインターコネクトで接続される．

ジョブ運用においては，一定数の計算ノードにつき 1 台の Burst Buffer ノードが割り当たった計算ノードグループを構成し，ジョブは割り当てられた計算ノードグループ内の Burst Buffer ノードを利用する．LLIO では，ジョブに割り当てられた計算ノードグループの SSD を束ねて，ノード内ローカル領域，ジョブ内ローカル領域，キャッシュ領域を構成する．

3.1.2 ソフトウェアスタック

LLIO のソフトウェアスタックを図 2 に示す．3 種類の領域は，別々のファイルシステムとしてマウントされ，アプリケーションは POSIX API を利用して各領域にアクセスする．LLIO Client モジュールは，アクセスする領域や I/O アクセス，メタアクセス種別に応じて，Burst Buffer ノードへ送信するリクエストを作成する．LLIO RPC モジュールは，LLIO のファイルアクセスのリクエストを生成する．ノード間の通信処理には，Lustre Network(LNET)モジュールを利用する．Lustre[13]は，スーパーコンピュータで広く使われているファイルシステムであり，2018 年 6 月の Top 500 の 10 位以内のシステムでも過半数が利用している．LLIO では，共有ファイルシステムが Lustre である前提を置き，LNET のインターフェースを利用して通信を行っている．

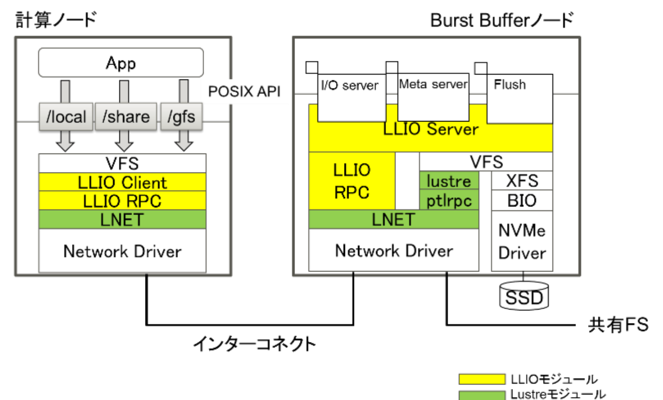

図 2 LLIO のソフトウェアスタック

Burst Buffer ノードにおいて，クライアントから届いたリクエストは LLIO RPC モジュールでデコードされ， Meta server プロセスおよび I/O server プロセスがリクエスト種別に応じた処理を行う．また，Flush プロセスは，リクエスト処理とは非同期にキャッシュ領域のデータを共有ファイルシステムに書き出す．これらのプロセスの処理は，LLIO Server モジュールに実装されている．LLIO は，3 種類の領域を 1 つの SSD で実現するため，それぞれの容量管理が必要となる．XFS には，ディレクトリ単位で容量の制限を設定できる project quota 機能を有する．そのため，LLIO では SSD のデバイス管理に XFS を利用する．

3.1.3 計算ノードのリクエスト発行

計算ノードが発行するリクエストには，メタリクエストおよび I/O リクエストが存在する．ジョブが複数の計算ノードグループに跨る場合，複数の Burst Buffer ノードを利用することになるが，それぞれのリクエストは以下のように各 Burst Buffer ノードに発行される．

(1) メタリクエスト

リクエストが特定の Burst Buffer ノードに集中し，高負荷状態となることを避けるため，計算ノードが属する計算ノードグループの Burst Buffer ノードに対してメタリクエストを発行する．

(2) I/O リクエスト

ノード内ローカル領域は，ノード間で共有しないファイルを置くのに適している．このようなファイルは，計算を行うプロセスが一時ファイルとして，同じタイミングで作成することが多い．そのため，各ノード間の I/O 性能の平準化を行うために，I/O リクエストは計算ノードが属する計算ノードグループの Burst Buffer に発行される．

ⓒ2018 Information Processing Society of Japan

ジョブ内ローカル領域，キャッシュ領域においては，I/O性能向上のため，ファイルデータを一定の転送サイズで分割し複数の Burst Buffer ノードにストライプ配置する．配置先は，ファイルの inode 番号及びファイル内のオフセットを元にクライアントが計算を行うことで, 一意に決める．そして，I/O リクエストは配置先の Burst Buffer に発行される．

2.2 で述べたように，スーパーコンピュータでは，プロセス多重度が増えた場合のスループットのみならず，単一プロセスの I/O 性能も重要となる．そのため，計算ノードでは，プロセスが一度に発行する READ/WRITE のサイズが転送サイズを超える場合，複数の I/O リクエストが生成され，並列で Burst Buffer ノードにリクエストを発行する．これにより, データ処理が並列化され, 単一プロセスの I/O 性能を高めることができる．

3.1.4 各プロセスの処理概要

本項では，Burst Buffer ノードにおいて，Meta server プロセス，I/O server プロセス，Flush プロセスが行う処理について述べる．

(1) Meta server プロセス

Meta server プロセスは，計算ノードが発行したメタリクエストを処理するプロセスであり，以下に示す方法で 3 種類の異なる名前空間を実現する．

ノード内ローカル領域は, 計算ノード毎に XFS 上の異なるディレクトリをルートディレクトリとして見せることで, 実現する．Burst Buffer ノード内では，計算ノードの IP アドレスとディレクトリの対応を管理しており，Meta server プロセスは, メタリクエストを発行した計算ノードの IP アドレスによってディレクトリを切り替える．

ジョブ内ローカル領域は，共有ファイルシステムにジョブ毎に異なるディレクトリを作成し，それをルートディレクトリとして見せることで，実現する．計算ノードが発行するメタリクエストには，ジョブ ID が付加されており，Meta server プロセスは，ジョブ ID を元にディレクトリを切り替える．

キャッシュ領域の名前空間は，共有ファイルシステムと透過である．そのため，Meta server プロセスでは，計算ノードのキャッシュ領域に対するメタリクエストの場合，共有ファイルシステムにリクエストをパススルーする．

(2) I/O server プロセス

I/O server プロセスでは，各領域に対する I/O リクエストを以下の方法で処理する．

ノード内ローカル領域とジョブ内ローカル領域では, I/O server プロセスは, 計算ノードの I/O リクエストに応じて, SSD の READ/WRITE, および計算ノードと Burst Buffer ノード間のデータ転送を行う．

キャッシュ領域では，I/O server プロセスは，WRITE リクエスト，または，SSD 上に存在するデータへの READ リクエストの場合，SSD の READ/WRITE, および計算ノードと Burst Buffer ノード間のデータ転送を行う．一方，SSD 上に READ するデータが存在しない場合，共有ファイルシステムからデータを読込み, SSD にキャッシングしたのちに，計算ノードに転送する．

(3) Flush プロセス

Burst Buffer ノードでは，キャッシュ領域に書き込まれたデータが共有ファイルシステムに書出し済みであるか否かの情報を管理している．Flush プロセスは，この情報をもとに，計算ノードからのリクエスト処理とは非同期にキャッシュ領域のデータを共有ファイルシステムに書出す.

Flush プロセスによる非同期書出し処理と I/O server プロセスのリクエスト処理によるデータ処理が同時に発生すると, リクエスト処理性能が低下してしまう. そのため, LLIO では，Flush プロセスによる非同期書き出し処理は，I/O server プロセスによるリクエスト処理が行われていないときに非同期書き出し処理を実行する．スーパーコンピュータのアプリケーションは，計算を行うフェーズと計算結果を書き出すフェーズが明確に分かれている場合が多いため, 計算ノードから I/O リクエストが来ず，非同期書き出しを実行できる時間帯が定期的に訪れると考えられる．

3.2 省資源型高速データ処理の実現

3.1.3 で述べた I/O server プロセス，Flush プロセスが行うデータ処理を限られたリソースで高速に行うためには，データ処理に掛かる CPU 負荷およびメモリ消費量を抑える必要がある．そのためのアプローチを以下で述べる．

(1) データ処理のデバイスへのオフロード

通信処理，SSD の READ/WRITE で発生する CPU 負荷を減らすために，LLIO では，RDMA，DMA を利用して，これらの処理をデバイスにオフロードする．

計算ノード上のバッファと Burst Buffer ノードの転送バッファ間の RDMA 転送は，計算ノードが LNET インターフェースを呼び出すことで実行する．

SSD デバイスと転送バッファ間の DMA 転送は，I/O server プロセスおよび Flush プロセスが VFS 経由で XFS に Direct I/O を発行することで実行する．

Burst Buffer ノードと共有ファイルシステム間の RDMA 転送は, I/O server プロセスおよび Flush プロセスが VFS 経由で共有ファイルシステムに Direct I/O を発行することで実行する. 共有ファイルシステムへの Direct I/O の延長で, LNET インターフェースを通じて RDMA 転送が行われる.

(2) 固定長の転送バッファを使ったデータ処理

RDMA および DMA を使用してデータ処理を行う既存のファイルシステムとして Lustre がある．Lustre はデータ転送用のバッファにページキャッシュを利用しているが，ページキャッシュはファイル毎，ファイル内のオフセット毎に異なるため，限られたメモリリソース下では，ページキャッシュ獲得と解放が頻発し CPU 負荷が上昇するため, 性

能劣化を引き起こす． LLIO では，ファイルやファイル内のオフセットに依らない固定長の転送バッファを確保し，データ処理に用いることで，ページキャッシュ獲得と解放の発生を抑止し，CPU 負荷の上昇を抑える．

LLIO では，SSD の READ/WRITE および共有ファイルシステムのデータ転送を XFS および Lustre クライアントへの Direct I/O で実施する．Direct I/O はバッファとしてユーザ空間のメモリアドレスを必要とするため，I/O server プロセスと Flush プロセスはユーザプロセスとして起動し，ユーザ空間にマッピングされたメモリ領域を転送バッファとして用いる．

(1)，(2)で述べた通り，LLIO でのデータ処理は，固定長の転送バッファと RDMA と DMA を利用して行うため，データ処理の過程でメモリコピーは発生せず，CPU 負荷，メモリ消費量の増加を抑える事ができる．その結果，限られたリソースでより多くのデータ処理を実行することができる．

3.3 I/O 処理ノイズの削減

Burst Buffer ノード上での I/O 処理によって生じる，計算を実行しているコアに対するデバイスからの割込みや OS によるスケジューリングを削減するために，計算用のコアと計算以外の処理を行うシステム処理用のコアを分離し，以下の方法でシステムタスク，割込み処理をシステム処理用コアにバインドする．

- システムタスクのシステム用コアへのバインドは，/sys/fs/cgroup/cpu/tasks に存在する全プロセスについて，taskset コマンドを使い，システム用コアに CPU アフィニティを設定することで行う．一部のカーネルスレッドについては，アフィニティを設定できないものがあった．これらのカーネルスレッドは，計算用コアでも動く状態で次章の評価を行った．

表 2. ブートパラメータの設定

パラメータ名	説明と指定した値
isolcpus	指定された CPU へのプロセスのスケジューリングを行わない．計算用コアを指定する．
nohz_full	指定された CPU を full tickless モードにする．CPU のランキューに存在するタスクが 0 または 1 つの時, タイマ tick 間隔を 1 秒に延長する．計算用コアを指定する．

表 3 カーネルパラメータの設定

パラメータ名	説明と指定した値
watchdog	各コアに対して定期的に上がる，ウォッチドッグのための NMI 割込みを有効/無効を切り替えるパラメータである．0 を指定し無効化する．

- 割込み処理のシステム用コアへのバインドは，/proc/irq/<IRQ 番号>/smp_affinity の値をシステム用コアの CPU マスクに設定することで行う. IRQ 番号 0 番のタイマ割り込みと IRQ 番号 2 番の NMI についてはシステム用コアへのバインドは不可能である．これらの割り込みについて，アプリケーションに与える影響を低減するために表 2，表 3 に示す設定を行う．

4. LLIO の評価

4.1 評価環境

本論文で用いた評価環境を表 4 に示す．Burst Buffer の転送バッファサイズ，および I/O server のプロセス数は，

表 4 評価環境

	Burst Buffer ノード	計算ノード
NUMA 構成	4 NUMA ノード	2 NUMA ノード
CPU	Intel® Xeon® CPU E5-4620 0@2.20GHz 8 物理コア × 4	Intel® Xeon® CPU E5-2632 0@3.30GHz 8 物理コア × 2
メモリ	16GB×4	32GB×2
インターコネクト	InfiniBand FDR ※NUMA ノード 1 に接続	InfiniBand FDR
SSD	Intel P3608 READ : 5.0GB/s, WRITE: 2.0GB/s ※NUMA ノード 1 に接続	—
OS	Red Hat Enterprise Linux 7.2	Red Hat Enterprise Linux 7.2
システム用コア	各 NUMA ノードの先頭コア	—
I/O server プロセス数	16	—
転送バッファサイズ	1MiB	—
ノード数	1	1, 2

ⓒ2018 Information Processing Society of Japan

表 5 IOR のパラメータ

パラメータ名	値
I/O サイズ	1MiB
ファイルサイズ	1GiB
クライアント多重度	1, 2, 4, 8, 16, 32
繰り返し回数	10
その他のパラメータ	プロセス別ファイル シーケンシャルアクセス
クライアント台数	1, 2

LNET の性能および仕様を元に決定した．LNET の 1 回当たりの RDMA 転送長は 1MiB であったため，I/O server プロセス当たりの転送バッファサイズは 1MiB とした．また，LNET の通信性能を測定したところ，多重度が 16 の時，RDMA READ/WRITE の性能が本評価で用いた SSD のデバイス性能を超えた．そのため，I/O server プロセス数は 16 とした．

評価の中では，ジョブ内ローカル領域に対して IOR を実行することで I/O を発行した．

4.2 省資源型高速データ処理の評価

限られたリソースで高い I/O 性能が実現されていることの評価として，以下を確認した．

- 1 コアで SSD のデバイス性能が引き出せること
- LLIO の I/O 処理中に転送バッファとしてページキャッシュが獲得されないこと

上記を確認するために，最初に，IOR を実行し SSD のデバイス性能が出ることを確認した．次に，ピーク性能が出ていた IOR の条件下で CPU 消費率を評価した．最後に，CPU 消費率の評価と同様の条件で IOR を実行し，転送バッファとしてのページキャッシュが確保されていないことを確認した．

4.2.1 I/O 性能の評価

本評価で実行した IOR のパラメータを表 5 に示す．計算ノード数が 2 台で，クライアント多重度が 2 以上の場合は，各ノードの多重度は均等にした．

図 3 に，READ/WRITE をそれぞれ 10 回計測したときの平均性能を示す．実線はクライアントノード 1 台で IOR を実行した結果，破線は 2 台で IOR を実行した結果である．

クライアントノード数に依らず，多重度が上がると平均性能が SSD のデバイス性能に近づくことが分かった．また，クライアントノードが 1 台と 2 台の場合それぞれについての最大性能を表 6 に示す．クライアントノードが 1 台，2 台両方の場合において SSD のデバイス性能を引き出すことができた．クライアント台数が 1 台の場合は多重度が 32 の時，2 台の場合は多重度が 16 の時に，READ/WRITE 共に最大性能に到達した．

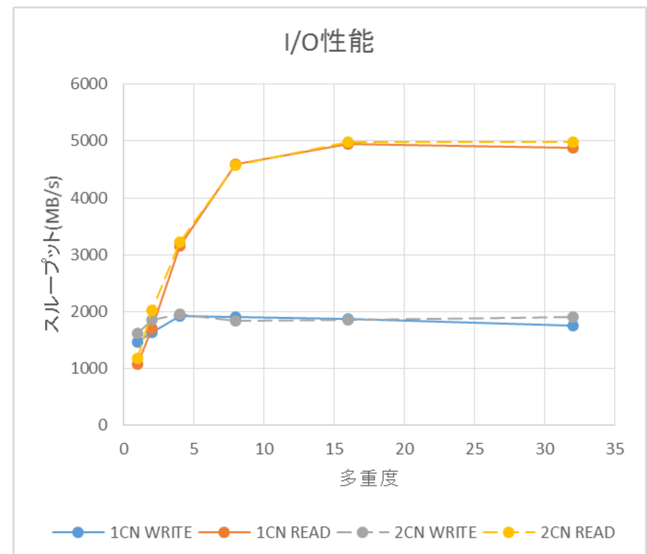

図 3 多重度を変化させたときの平均 I/O 性能

表 6 クライアントノード数別の最大 I/O 性能

	クライアントノード数	
	1 ノード	2 ノード
READ	5007MB/s	5064MB/s
WRITE	2008MB/s	2020MB/s

表 7 I/O 処理中の CPU 消費率

	CPU 消費率
READ	94.84%
WRITE	38.62%

表 8 I/O 前後の最大メモリ消費量

	メモリ消費量
I/O 処理前	27,107,328 byte
WRITE	27,099,136 byte
READ	27,095,040 byte

4.2.2 CPU 消費率の評価

本節の評価は，クライアントノードを 2 台用いて実施した．クライアントノードが 2 台の場合，4.2.1 では，多重度 16 の時に READ/WRITE 性能が最大となった．そのため，表 5 における多重度 16 の条件で I/O を発行し，Burst Buffer ノードの CPU 消費率の評価を行った．なお，CPU 消費率を測定するための十分な時間を確保するため，1 プロセスが READ/WRITE するファイルサイズは 10GiB とした．

CPU 消費率は，IOR 実行中に sar コマンドを用いて Burst Buffer ノードのシステム全体の CPU 消費率を 30 秒間測定し，平均値を算出した(表 7)．CPU 消費率 100%は，1 コア使い切ることを表している．READ/WRITE 共に 1 コアを使い切ることなくデバイス性能を出すことができた．

4.2.3 メモリ消費量の評価

前項と同様のパラメータで I/O を発行し，Burst Buffer ノードにおける LLIO の I/O 処理中のメモリ消費量の増減を評価した．

メモリ消費量の変化を調査するために，cgroups の memory サブシステムにコントロールグループを作成し，LLIO で生成した全てのプロセス，LNET が生成した kiblnd_sd プロセスを当該コントロールグループに追加した．メモリ消費量の測定は，memory.max_usage_in_bytes を 1 秒間隔で参照することで行った．

表 8に測定で得られたI/O処理前のメモリ消費量, READ 処理中，WRITE 処理中の最大メモリ消費量を示す．測定中に発行した合計 I/O サイズは，10GiB × 16 プロセス = 160GiB であったが，I/O 処理前，READ/WRITE 中で memory.max_usage_in_bytes の値がほぼ一定であり，転送バッファとしてのページキャッシュは獲得されていないことが分かった．

4.2.4 考察

今回の評価環境と評価パラメータにおいては，1 コアで READ/WRITE 共にデバイス性能を出し切ることができた．また，I/O 処理中にページキャッシュ獲得が発生していないことも確認できた．

一方で，今回の評価は限定的であり，デバイス性能が高くなった場合，また，より小さい I/O サイズによるファイルアクセスが発行された場合には，ソフトウェアオーバーヘッドがデバイス処理に対して相対的に大きくなり，1 コアでデバイス性能を出すことは難しくなる．CPU 消費率を下げるための改善点を見出すために，4.2.2 でほぼ 1 コアを使い切っていた READ 処理について，perf を使い CPU 消費率の内訳を解析した．解析の結果，ページ数に比例した処理の割合が約 20%と大きくなっていた．

ページ数に比例した処理を減らすには，ページサイズを大きくすることが考えられる．評価環境はページサイズが 4KiB であったが，x86_64 は 2MiB の HUGE PAGE をサポートしており, aarch64 では, ページサイズが 4KiB, 16KiB, 64KiB から選択可能である．これらの大きいページを転送バッファとして用いることで，ページ数に比例した処理を削減できる．例えば，HUGE PAGE を使用することでページ数に比例した処理に掛かる CPU 消費率はほぼ 0%となるため，READ の CPU 消費率は，75%程度になると予想される．そのため，1.3 倍程度高速なデバイスでも 1 コアでデバイス性能が出せる可能性が出てくる．

一方，小サイズの I/O はページ数削減の恩恵は受けにくい．そのため，小サイズの I/O はクライアントのページキャッシュに溜め，サーバに対して 1MiB などの大きなサイズの I/O リクエストを発行する，といった処理を行い小サイズの I/O リクエストを減らす必要がある．

表 9 FWQ で指定したパラメータ

オプション名	値	説明
-w（仕事量）	18	ノイズ解析の分解能を上げるため，一回の仕事の処理時間がタイマ割り込みに対して小さくなるように指定した
-n（測定回数）	2,000,000	周期の長いノイズの測定漏れを防ぐため，測定時間が十分程度になる測定回数を指定した
-t （スレッド数）	32	全コアで測定するように指定した．

4.3 I/O 処理ノイズの評価

本節では，I/O 処理中に生じる計算時間のばらつきを Fixed Work Quantum(FWQ)[14]を用いて測定し，大規模並列環境でもアプリケーションの性能劣化が小さく抑えられることを確認した．

4.3.1 FWQ を使った性能劣化の評価方法

FWQ は OS が計算に与えるノイズを調査するのに用いられるツールである．FWQ では，一定の仕事量の処理を繰り返し行い，その処理時間を測定する．そして，処理時間のばらつきからノイズ原因の調査，アプリケーションの性能劣化の評価が可能である．

本評価では，LLIO の I/O 処理が行われている状態で，FWQ によりノイズを測定し，以下の方針で，I/O 処理がアプリケーションに与える影響が小さいことを確認した．

- 平均ノイズ率が十分に小さく，ノイズによる計算時間のばらつきはほとんど生じないこと
- 最大ノイズ長を記録したノイズの発生間隔を調査し，Tsafrir の式[15]を用いて，大規模並列環境において当該ノイズによる性能劣化が小さくなること

平均ノイズ率とは，単位時間当たりに計算時間がどの程度遅延するかの平均を表した数値であり，処理が遅延した時間の平均を，理論上の計算時間で割ることで得られる．「京」および FX100 の実績[12]から5.0×10^{-5}を下回れば，平均ノイズ率が十分に小さいと判断した．

最大ノイズ長は，測定の中で最も大きかった遅延を表す．Tsafrir の式は，ノイズによる計算時間のばらつきから，並列環境における性能劣化を算出する式であり，以下で表される．

$$slowdown = 1 + \left(1 - \mathrm{e}^{-\frac{NG}{T}}\right)\frac{D}{G}$$

各パラメータは，以下の意味を持つ．

- N: 並列度

  アプリケーションの並列度を表す．

- G: 計算フェーズ時間
  ノイズが入らない場合のアプリケーションの同期と同期の間の計算時間を表す.
- T: ノイズ間隔
  ノイズの発生間隔を表す.
- D: ノイズ長
  ノイズによって起きる計算の遅延を表す.

T および D は，FWQ の測定結果により求まる数値である．N および G は,「京」と同規模のシステムを想定して，$N = 100{,}000$，$G = 1\text{ms}$とし，最大ノイズ長となったノイズによる性能劣化が 5%以内となることを確認した．

FWQ で指定したパラメータの値は表 9のとおりである．I/O 処理は，IOR を 4.2.2 と同様に，最大性能を記録した条件で実行し発生させた．

4.3.2 評価結果

表 10 に計算コアで測定された平均ノイズ率，最大ノイズ長の最大値と最大ノイズ長の発生間隔を示す．平均ノイズ率は，全計算用コアにおいて，5.0×10^{-5}を大きく下回っているため，平均的なノイズによる性能劣化は非常に小さいことわかった．

また，Tsafrir の式に「京」規模とした仮定したパラメータ$N = 100{,}000$，$G = 1\text{ms}$および，測定の結果得られた$D = 41\mu\text{s}, T = 1\text{s}$を代入し計算したところ，性能劣化は 4.1%となり，5%以内に収まっていることが確認できた．

4.3.3 考察

京と同様の 10 万並列規模においては，I/O 処理がある場合でもアプリケーションの性能劣化は 5%未満に抑えられ，計算コアとシステム処理用コアの分離が有効であることが分かった．本測定では，最大ノイズ長となったノイズの発生間隔は 1 秒であったが，これは，計算コアに対するタイマ割り込みであると推測される．表 10 で示した通り，計算コアに対して nohz_full および isolcpus を指定しており，FWQ の実行中には，各計算コアには FWQ のスレッドとアフィニティ設定が不可能なカーネルスレッドのみがスケジューリングされる可能性があった．しかし，カーネルスレッドは待ち状態になっており，計算コアのランキューには FWQ のスレッドのみが存在していたために，タイマ割り込みの間隔が 1 秒ごとに遅延されたと考えられる．タイマ割込みは I/O 処理が無い場合でも生じるため，本評価からは，LLIO の I/O 処理がアプリケーションの外乱となることは無かったと言える．

今回測定に使った FWQ では，変数のカウントアップの処理時間を測定していた．変数のカウントアップは，処理中にメモリアクセスが生じないため，システム処理用コアと計算コアの分離により処理時間のばらつきを小さく抑えることができた．しかし，計算中にはメモリアクセスを伴うことが一般的であり, 仮にコアを分けたとしても LLC を共有しているため, I/O 処理の影響が現れる可能性がある．

表 10 ノイズ測定結果

項目	値
平均ノイズ率	1.09×10^{-5}
最大ノイズ長	41μs
最大ノイズ長となったノイズの発生間隔	約 1s

メモリアクセスを伴う計算に与える影響の評価は，今後，実アプリケーション等を用いて行っていく．

5. まとめ

本論文では，計算ノード Burst Buffer 実現のための課題として, 限られたリソースで高い I/O 性能を実現すること，I/O 処理がアプリケーションに与える性能影響を低く抑えることを挙げ，これらの課題を改善するデータ処理方式について検討と評価を行った．

(1) 省資源型高速データ処理の実現

固定長バッファを使った RDMA，DMA によるデータ処理を行うことで，1 コアでデバイス性能を出し切ることが可能であり，I/O 処理中のページキャッシュ獲得によるメモリ消費が発生しないことが確認できた．

(2) I/O 処理ノイズの削減

計算コアとシステム処理用コアを分けることで，システム処理用コアで実行されている I/O 処理が FWQ の計算時間に影響を及ぼさないことが確認できた．また,「京」と同規模の並列環境下においても性能劣化が 5%以内に抑えられることが分かった．

今後は，転送バッファのページサイズを大きくした場合の CPU 消費率改善の検証，および，より大規模な環境でのメタ性能を含めた性能評価，I/O 処理が実アプリケーションに与える性能影響の評価を実施していく予定である．

謝辞　本論文の一部は，文部科学省「特定先端大型研究施設運営費等補助金（次世代超高速電子計算機システムの開発・整備等 ）」で実施された内容に基づくものである．

参考文献

[1] “Summit”, https://www.olcf.ornl.gov/summit/, (参照 2018-06-26)
[2] “Cori-nersc”, http://www.nersc.gov/users/computational-systems/cori/, (参照 2018-06-26)
[3] “ABCI: AI Bridging Cloud Infrastructure”, https://abci.ai/, (参照 2018-06-26)
[4] “Oakforest-PACS | 東京大学情報基盤センター スーパーコンピューティング部門”, https://www.cc.u-tokyo.ac.jp/guide/hpc/ofp/, (参照 2018-06-26)
[5] “TSUBAME3 | [GSIC]東京工業大学学術国際情報センター”, http://www.gsic.titech.ac.jp/tsubame3, (参照 2018-06-26)
[6] “IME® FLASH-NATIVE DATA CACHE | DDN - DDN Storage”, https://www.ddn.com/products/ime-flash-native-data-cache/, (参照 2018-06-26)

ⓒ2018 Information Processing Society of Japan

[7] “Cray Supercomputer, XC Series Supercomputers - DataWarp - Cray”, https://www.cray.com/products/computing/xc-series?tab=datawarp, (参照 2018-06-26)

[8] T. Wang, K. Mohror, A. Moody, K. Sato and W. Yu, "An Ephemeral Burst-Buffer File System for Scientific Applications," SC16: International Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, UT, 2016, pp. 807-818.

[9] Zhao, D., Zhang, Z., Zhou, X., Li, T., Wang, K., Kimpe, D., Carns, P., Ross, R., Raicu, & I. "FusionFS: Toward supporting data-intensive scientific applications on extreme-scale high-performance computing systems," 2014 IEEE International Conference on Big Data (Big Data), Washington, DC, 2014, pp. 61-70.

[10] “Exploiting Node-Local, Non-Volatile Memory (NVM) – Oak Ridge Leadership Computing Facility “, https://www.olcf.ornl.gov/olcf-resources/rd-project/exploiting-node-local-non-volatile-memory-nvm/, (参照 2018-06-26)

[11] “BeeGFS Wiki: BeeOND™: BeeGFS On Demand”, https://www.beegfs.io/wiki/BeeOND, (参照 2018-06-29)

[12] “FUJITSU Supercomputer PRIMEHPC FX100 先進のソフトウェア”, http://www.fujitsu.com/downloads/JP/archive/imgjp/jhpc/primehpc/primehpc-fx100-soft-ja.pdf, (参照 2018-06-29)

[13] “Lustre”, http://lustre.org/, (参照 2018-06-26)

[14] “ASC Sequoia Benchmark Codes”, https://asc.llnl.gov/sequoia/benchmarks/, (参照 2018-06-26)

[15] Tsafrir, Dan. "The context-switch overhead inflicted by hardware interrupts (and the enigma of do-nothing loops)." Proceedings of the 2007 workshop on Experimental computer science. ACM, 2007.

ⓒ2018 Information Processing Society of Japan