Vol.2018-HPC-165 No.6
2018/7/30

ファイルシステムの利用高度化に向けたスパコンセンター間での情報共有の取り組み

辻田　祐一[1,a)]　中川 剛史[2]　板倉　憲一[2]　宇野 篤也[1]

概要：演算処理や MPI 通信などに比べ, ファイル I/O の最適化事例は少ない．その要因としては，スーパーコンピュータ（以下, スパコン）を運用する側からの並列ファイルシステム（以下,PFS）の特性に関する情報提供やユーザの PFS に対する理解が不十分であること，さらには MPI-IO に代表されるファイル I/O ライブラリの実装が PFS を含むシステムの特性を十分に活かせていない，などの問題が挙げられる．このような背景に鑑み, スパコンセンター間で PFS を含むシステムの性能情報の共有並びに情報提供を行うことは, ユーザの利用技術およびシステム運用の高度化に有益であると考えている．この考えに基づいて，本稿では海洋研究開発機構が運用する地球シミュレータと，理化学研究所が運用するスーパーコンピュータ「京」において, 情報共有とユーザへの情報提供を目的に, MPI-IO を中心とした性能比較等を行った．評価結果から，I/O パターン依存の挙動など，各々の PFS の特性の違いなどが明確になり, 有益な情報共有の可能性を確認した．

キーワード：情報共有, 地球シミュレータ, ScaTeFS, スーパーコンピュータ「京」, FEFS, MPI-IO

1. はじめに

近年の並列計算の規模増大に伴い，スパコンのシステムの高度化・複雑化が進んでいる．またユーザのニーズも益々多様化しており, 一つのスパコンセンター内で全ての運用技術の高度化を進めることは, 年々困難な状況になってきている．その状況を踏まえ, スパコンセンター間で可能な範囲で以下のような項目の情報共有の取り組みが必要と考えている．

- スパコンに関するハードウェア・ソフトウェアの機能・性能情報等
- ユーザの利用実態と高度利用技術
- 運用技術

様々な要素を備えるスパコンの中でも, 特に PFS や MPI-IO [1] などの入出力ライブラリに関する情報共有とユーザの利用技術の高度化支援が急務と考えている．計算処理や MPI [2] を用いたプロセス間通信などに対しては，使用する計算機に合わせた最適化が積極的に行われているが，ファイル I/O に対してシステム毎に最適化を行っている事例はそう多くない．間違った使い方を継続しているケースも散見され, 場合によってはシステムの運用に大きな影響を及ぼすケースも確認されている [3].

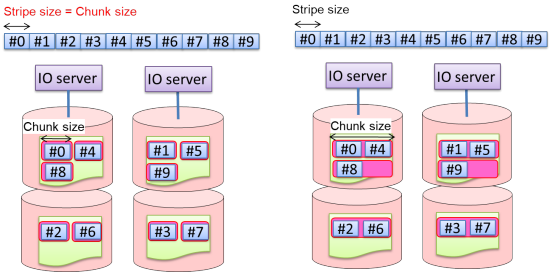
そのような状況を招く要因の一つとしては，PFS の特性などを十分に理解せずに利用していることが挙げられるが，それ以外にもシステムを運用する側からの PFS に関する特性や効果的な利用方法などの情報提供が不十分である点や,MPI-IO の実装において, 配下の PFS やプロセス配置に対する最適化が不十分であることも挙げられる．

特に MPI-IO は，PnetCDF [4] や HDF5 [5] などのアプリケーション向け I/O ライブラリの並列 I/O 機能も担当しており，MPI-IO の高速化・高度化は重要な課題の一つである．MPI-IO には複数のプロセス間で共有ファイルに対する I/O を集団的に行う集団型 MPI-IO 関数があり，PnetCDF や HDF5 でも多用されている．集団型 MPI-IO 関数は, 配下の PFS の仕様等に依存して実装内部の動作が異なり, I/O パターンによっては意図せずに性能低下を招いていることもある．

そこで, 我々はスパコンセンター間の情報共有の有効性の検証の一環として, MPI-IO を中心としたファイル I/O の高度利用・高度運用に向けて, PFS の性能・特性も含めた情報共有の有効性を海洋研究開発機構が運用する地球シミュレータ（以下,ES） [6] と理化学研究所が運用するスーパーコンピュータ「京」（以下,「京」） [7] を用いて検証した．各々のスパコンの PFS である ScaTeFS 並びに FEFS [8]

[1] 理化学研究所　計算科学研究センター
[2] 海洋研究開発機構 地球情報基盤センター
[a)] yuichi.tsujita@riken.jp

© 2018 Information Processing Society of Japan



(a) “ノンストライピング” のレイアウト例　(b) “ストライピング” のレイアウト例

図 1: ScaTeFS におけるストライプ機構

での MPI-IO を中心とした性能評価を行い, 評価結果から各々の PFS の特性に起因する性能低下のパターンなどの分析を行うと共に，各システムが提供する MPI-IO 実装自体の問題点などにも言及し，性能改善の可能性に関しても検討を行った.

以下, 第 2 章では,ES および「京」のシステム構成について概説し, 第 3 章において,MPI-IO 実装における集団型 MPI-IO 関数の処理フロー並びに IOR ベンチマークを例に取った I/O パターンに依存した挙動に関して説明する. 次に, 今回行った性能評価に関して第 4 章で報告する. 関連研究について第 5 章で言及した後に, 第 6 章において本稿のまとめを行う.

2. 地球シミュレータおよびスーパーコンピュータ「京」のシステム構成

2.1 地球シミュレータ

NEC の SX-ACE を中心に構成された ES は,5,120 台の計算ノードを有している. ノード間は Internode Crossbar Switch(IXS) により接続されており, ノード間の理論帯域は 4 GiB/s× 双方向となっている.

ファイルシステムには NEC が開発した PFS である ScaTeFS を採用している. ScaTeFS は複数の IO サーバにメタデータおよび実データの両方を分散して管理しており, **図 1** に示すように, 各 IO サーバには分散されたデータを格納する実ファイルが配置される. なお, 各実ファイルの大きさは chunk size と呼んでいる. chunk size は分散されたデータ断片の大きさ（ストライプサイズ）の整数倍である必要があり,chunk size とストライプサイズが等しい場合を “ノンストライピング”, それ以外を” ストライピング” と呼んでいる. ストライプカウントは当該ボリュームの IO ターゲット数あるいはファイルサイズをストライプサイズで割った値のいずれか小さい方となる.

ES の ScaTeFS は**図 2** に示すように, RAID-6 のストレージで構成されており, 128 ノード単位で割り当てている work 領域と, システム全体で共有される data 領域の二種類を提供している. 前者はシステム全体で 40 個のボリュームを有しており, ボリュームあたり 2 台の IO サーバが配置されている. ジョブ実行時にランク毎に一意的なディレクトリが用意され, 干渉の少ないローカル I/O も可能である. また,MPI-IO によるファイル I/O も利用できるが, 空いている計算ノードを通信経路が最短になるように順に割り当ててゆくスケジューリングポリシーのため, ノード間で異なる work 領域間を跨ぐ場合には, 全プロセスによる集団型 MPI-IO は行えない. 一方, 後者はフロントエンドサーバ群からもアクセスできるグローバルな領域であり, システム全体で 7 個のボリュームを有している. いずれの領域も計算ノード群と IO サーバ間は 10 Gbps Ethernet により接続されており, IO サーバから IO ターゲットとなるストレージ装置（図中の iStorage M300）間は FibreChannel で接続されている. なお, ジョブ間のファイル I/O における干渉緩和のために, data 領域と work 領域間は, ステージングによるデータコピー処理が行われる.

2.2 スーパーコンピュータ「京」

図 3(a) に「京」のファイルシステムを含むシステム構成を示す. 「京」では，計算ノードでのファイル I/O の性能を確保するために，ジョブ実行時に使用するファイルシステム（Local File System：以下，LFS）と，ユーザのプログラムやデータを格納するファイルシステム（Global File System：以下，GFS）で構成される 2 階層のファイルシステムを採用している. 前者はジョブ実行中の高速なファイル I/O 向け, 後者は大容量のデータ領域という位置付けになっており, ファイルシステムを構成するストレージは LFS および GFS に対し, それぞれ RAID-5 並びに RAID-6 を採用している. いずれの領域も富士通により Lustre [9] をベースに開発された FEFS を用いている. データのストライピング処理など基本的な機能は Lustre の機能を継承しており, 大規模運用向けの様々な拡張機能を有している.

「京」の計算ノードおよびファイル I/O 用のノード（以下,I/O ノード）の間は Tofu [10] と呼ばれるインターコネクトで接続されている. 図 3(b) に示すように, 計算ノードからは,Object Storage Server （OSS）が動作する I/O ノードを経由して, その先の LFS にアクセスできる. 一方,GFS は I/O ノードから InfiniBand（図 3(a) の Global I/O Network）により接続されている. ジョブ実行においては, 基本的に非同期型のステージング機構 [11, 12] により, GFS と LFS の間でファイルの転送処理を行っている. ステージング処理は他のジョブが実行されている最中に並行して行われるため，全体のジョブ実行効率の向上に大きく寄与している. また, ジョブ実行中の各ランクからのファイル I/O 効率を高めるために, ランク間で共有される領域に加え, ランク毎に独立にアクセス可能なランク番号ディレクトリと呼ばれる loopback ファイルシステムも提供し

© 2018 Information Processing Society of Japan

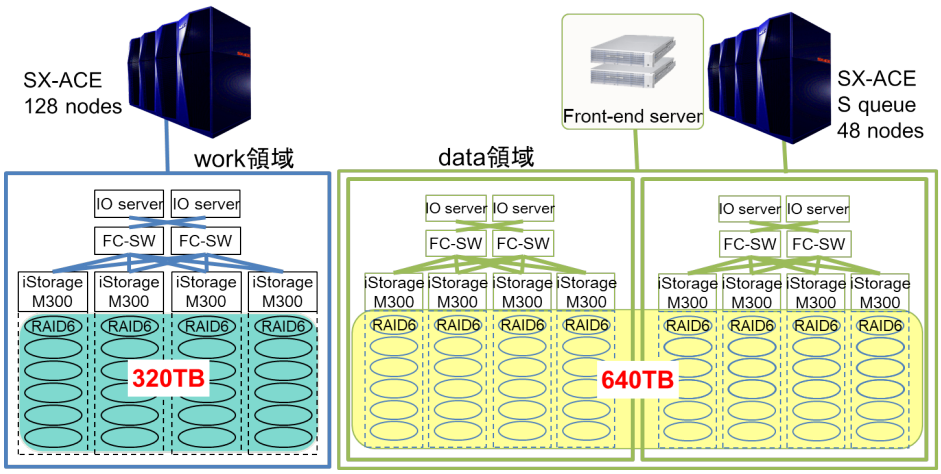
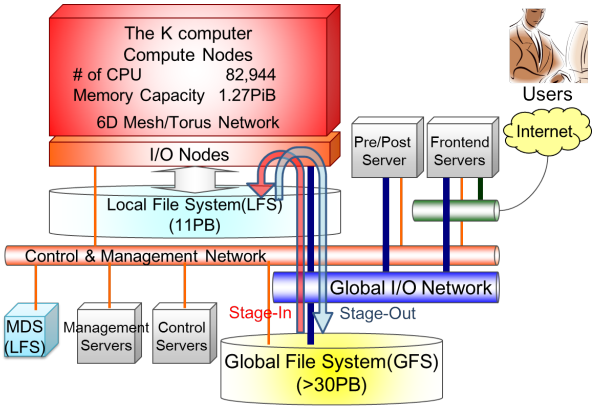
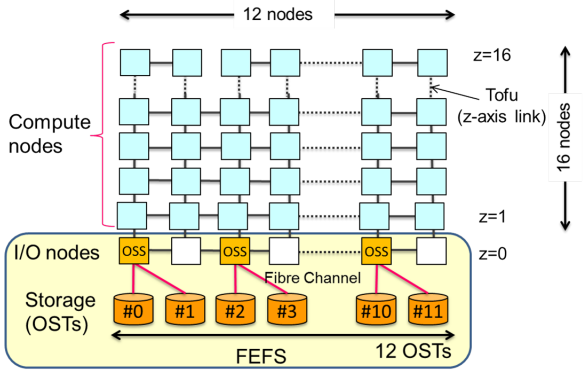


図 2: ES のファイルシステム構成



(a) 「京」のファイルシステムを含むシステム構成



(b) LFS と I/O ノードの構成の一例（2 × 3 × 32 の配置）

図 3: 「京」のファイルシステム構成

ている.

前述の ES と「京」の PFS 構成を比較すると,ES の work 領域は「京」の GFS の構成に近いもので, 計算ノード群との結合度は緩く, また故障時の弊害を最小限に留めるような構成を取っているが,ES の work 領域においてランク毎に用意されるディレクトリは,「京」の LFS に構築されるランク番号ディレクトリに近いものと言える. 一方,「京」の LFS では,I/O ノード群と計算ノード群との結合度が高く, かつ I/O ノードから FibreChannel を介してストレージにアクセスでき, 全計算ノードで一つのボリュームを構成していることからも, 高性能かつ高スケーラビリティを確保する性能重視な構成になっていると言える.

3. MPI-IO 実装

MPI-IO の代表的実装である ROMIO [13] は開発母体である MPICH [14] だけでなく,OpenMPI [15] の MPI-IO 機能の一つとしても利用されている. 今回評価試験等を行った ES および「京」共に、ROMIO をベースにした MPI-IO 実装を提供している.

ROMIO は MPI-IO インタフェース部を含むファイルシステム共通のレイヤの下に複数のファイルシステム用のドライバレイヤを有している. MPI-IO の関数群の中でも, 特に集団型 MPI-IO と呼ばれるものは PnetCDF や HDF5 における並列 I/O 機能実現にも重要な役割を担っている. 例えば気象関係のシミュレーションコードでは PnetCDF の利用が広まっており, MPI-IO の更なる利用拡大が今後も期待される. 以下, 集団型 MPI-IO における最適化実装である two-phase I/O [16]（以下,TP-IO）の実装状況や I/O パターンに依存した集団型 MPI-IO 関数内の挙動に関して説明する.

3.1 Two-Phase I/O 実装

複数のプロセス間で共有されたファイルに対して I/O を行う場合，問題となるのが飛び飛びのアクセスパターンである. **図 4** に 4 プロセス間で 2 次元配列データを分割して管理している状態でファイルに書き出しをする例を示す. 論理的に 2 次元配列をプロセス間でブロック分割しているが，共有ファイルに書き出す際のファイルビュー上では, 各々のプロセスが保有するデータの書き込み先が飛び飛びになっており，不連続なファイル I/O を各プロセスが行うと, 著しい性能低下を招く.

この問題を解消するために TP-IO が実装されている. **図 5** に上記の例における TP-IO の処理フローを示す. こ

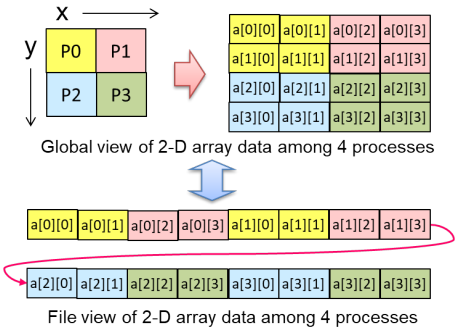
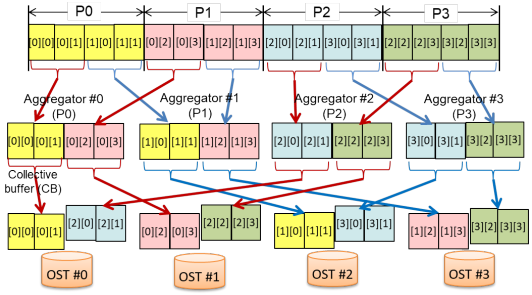
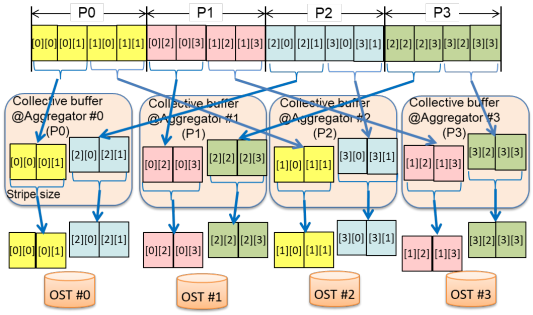


図 4: 2 次元配列データを 4 プロセス間で分割した際のファイルビューの例



(a) ファイルビュー指向



(b) ストライピングレイアウト指向

図 5: TP-IO での処理フロー

こで示した 2 つの処理フローのうち, 図 5(a) に示すファイルビュー指向は汎用的な実装であり, 今回の取り組みにより, ES および「京」共に, 集団型 MPI-IO はこの実装方法を踏襲していることを双方で情報共有している. この手法では, ファイルビュー上での各ユーザプロセスの非アクセス領域も含むデータ区間の先頭と最後尾の間がプロセス間でオーバーラップしない場合には, 各プロセスが自身の持つデータのファイル I/O をそのまま行い, そうでない場合にはこの図に示した手法を用いる.

TP-IO を開始するにあたり, ファイル I/O を行うプロセス（アグリゲータ）間でファイルビュー上で均等に分割し, 担当領域を割り当てる. 次に,Collective buffer（以下,CB）と呼ばれる有限の大きさのバッファを用い, ファイル I/O が連続になるようにアグリゲータ上のデータ並べ替えの通信とアグリゲータによるファイル I/O を交互に繰り返す. なお,ES および「京」のデフォルトの CB の大きさは, それぞれ 4 MiB 並びに 16 MiB になっているが, ユーザプログラム側で `cb_buffer_size` のキーにより `MPI_Info_set()` を用いて変更可能であり, 適宜性能チューニングが行える.

この手法の問題点としては,I/O パターンにもよるが, Lustre のようなストライピングアクセスを行うファイルシステムに対しては, 各アグリゲータから Object Storage Target（OST）群へのアクセス混雑が発生し, 性能向上が限定的になる点が挙げられる. これに対して図 5 のストライピングレイアウト指向の方法は, Lustre 向けの ADIO ドライバレイヤ [17] において採用されている手法であり, 各アグリゲータが特定の OST にアクセスするようにデータ並べ替えを行う点が特徴で, これにより前述の混雑を回避し, 性能向上を実現する. なお,ES の ScaTeFS や「京」の FEFS はストライピングアクセスを行う並列ファイルシステムであり, 特に後者は Lustre をベースにしているため, ストライピングレイアウト指向の方が性能向上できる可能性がある. 実際に「京」において EARTH on K [18] と名付けた ROMIO をベースとした最適化実装により性能向上の可能性を示しているが, 本稿ではシステム標準のベンダ提供の MPI-IO 実装による性能評価をベースに議論する.

ユーザ側で `MPI_Info_set()` を用いて設定変更を行う際に, いずれの手法にも関係するキーとして, 例えば強制的に TP-IO を実施させるか否かを指示する `romio_cb_{read/write}` があるが, 今回の情報共有により,ES では `cb_{read/write}` であることが判明している. 同様に他にもいくつか違いがあるものが分かっており, 情報共有と共に, 誤った使い方を防ぐためにユーザへの周知徹底の必要性を確認した.

3.2 I/O パターンに依存した集団型 MPI-IO の挙動

集団型 MPI-IO の使い方によっては, ファイルシステムの特性を活かせず十分な性能を発揮できていないケースが散見されている. その一例として Lustre のような PFS において IOR ベンチマークを用いた際の性能低下を招く事例を**図 6** に示す. この図に示した事例は IOR で良く用いられるファイル I/O パターンである. IOR のファイル I/O パターンには (1) 各プロセスが一度にアクセスするサイズ（XFER）, (2) 各プロセスの連続した分担領域の大きさであるブロック長（BLK）, 並びに (3) それらのブロックを結合した集合体の個数であるセグメント数の 3 種類がある. この図では, セグメント数が 1 のレイアウトの例を示しており, 各ランクのプロセスが共通のファイルに対してアクセス領域を分担している.

このレイアウトに対するファイル I/O に関して, ファイルビュー指向の TP-IO では, 前述の通り, ファイル I/O パターン分析により, 各ランクが独立に自身のデータのファイル I/O を行う. よって, 各ランクは, ファイルビュー

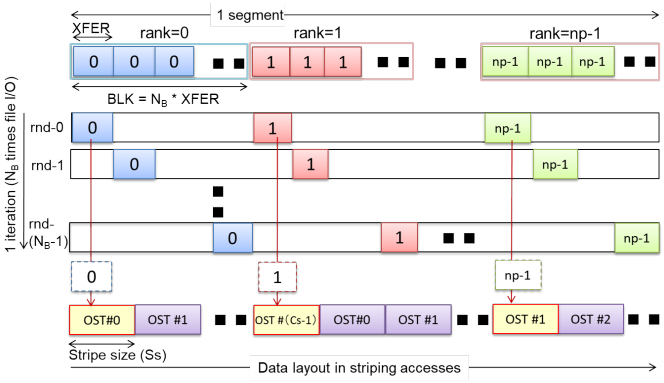


図 6: IOR ベンチにおける性能低下を招く I/O パターンの一例

上で BLK（$BLK = N_B \times XFER$）の大きさに達するまで,XFER 単位のファイル I/O を N_B 回だけ実施する．一方, 配下の PFS においてはストライプサイズ（以下,S_S）単位で OST 間をラウンドロビンでアクセスされる格好になっている．そのため,S_S およびストライプカウント（以下,C_S）と I/O パターンの組み合わせによっては, 各回のファイル I/O において一部の OST にアクセスが局所化してしまい, 全体の I/O 性能が向上しない．

なお, ファイルビュー指向の TP-IO では, 強制的に TP-IO の処理を実施させるオプションがあるが, この場合,CB 単位でアグリゲータにデータを並び替えて, 各アグリゲータにより CB 単位でファイル I/O を行う点が異なるだけで,PFS 上のデータ配置が決まっている以上, 特定の OST にアクセスが局所化することに変わりは無い．ファイル I/O の CB の大きさが S_S 以下, あるいは S_S 以上で, ファイル I/O の各回において全体でアクセスされる OST 数が多いほど性能が向上する可能性が高くなる．また, ストライピングレイアウト指向の TP-IO においては, S_S 単位でアグリゲータによってファイル I/O を行う点では異なるが, OST 群への局所化したアクセスでは同様であり, 特定のアグリゲータにデータ通信およびファイル I/O が集中し, 全体の性能向上が望めない．

よって,I/O 性能の向上の観点からは, このようなアクセスが局所化するような I/O パターンを避け, 出来るだけ多くの OST へアクセスする I/O パターンに変える必要があるが, 改変を支援するような情報提供の事例はほとんど見当たらない．その意味において, 性能向上を実現する, あるいは性能低下に陥りやすい I/O パターン等や性能情報を含むファイル I/O 全般の情報を提供する取り組みが必要であり, 複数のスパコンセンター間で情報を共有しながら情報公開に努めることはユーザにとって有用と考えられる．

4. 性能評価

前章で述べたような背景に鑑み,ES 及び「京」において, 双方のスパコンセンター間で情報共有を行いながら, 双方の PFS の特性を調べるために IOR による POSIX-I/O 並びに MPI-IO の性能評価を行った．本評価では IOR ver. 2.10.3 を用い, 双方とも 96 ノードの計算ノード上に, ノードあたり 4 プロセスの配置で計 384 プロセスを起動して評価を実施した．さらに, 必要に応じてノード数・プロセス数を変えるなどにより, システムの特性に関する検証試験も行った．IOR のパラメタのうち,BLK の大きさに関して, ES においては,ScaTeFS のデフォルトの chunk size 設定である 256 MiB を考慮し,”ノンストライピング”レイアウトとなる S_S の最大値である 256 MiB に設定した計測を基本とし, 補足的に chunk size を越えた長さにおける検証として, 512 MiB の設定でも行った．一方,「京」においては読み出し処理のクライアントキャッシュの影響を確認できる大きさとして 512 MiB の設定で試験を実施した．いずれもセグメント数を 1 として,XFER を変化させて計測した．PFS の設定については,S_S の最小値を 1 MiB とし, 最大値を XFER の最大サイズとした上で, POSIX-I/O および MPI-IO の 2 種類について, 以下のコマンドにより評価を行った．

- POSIX-I/O

```
./IOR -i 5 -a POSIX -k -m -w(-r) \
  -t ${XFER}m -b ${BLK}m -s 1 \
  -o ${fname_prefix} -d 0.1
```

- MPI-IO

```
./IOR -i 5 -a MPIIO -c -k -m \
  -U ${HINTS_FILE} -H -w(-r) -t ${XFER}m \
  -b ${BLK}m -s 1 -o ${fname_prefix} -d 0.1
```

ここで${*fname_prefix*} はファイル I/O 対象のファイル名である．また${*HINTS_FILE*} は,MPI_Info_set で設定するキーと値のセットを記載したファイルである．このコマンドにある”-w” と”-r” のオプションを切り替えて書き込みと読み出しを分けて実施しており, 各パラメタセット毎に,5 回の書き込み処理を行った後に, そこで生成されたファイルを用いて同じ回数の読み出しを行い, 平均値および最大・最小値を集計した．

4.1 ES における性能評価結果

ES のローカルファイルシステムである work 領域上での MPI-IO 性能を計測した．前述の通り, 当該領域は 128 台の計算ノード単位で 1 ボリュームを配置する格好のため, 同じボリュームを共有する 96 台の計算ノードに 384 プロセスを起動して性能を計測した．work 領域に対する設定は,chunk size はデフォルトの 256 MiB, S_S は”ストライピング”となる 1 MiB および 16 MiB 並びに”ノンストライピング”となる 256 MiB の計 3 パターンを用いた．work 領域の IO ターゲット数は 24 であり, C_S はこの 24 あるいはファイルサイズを S_S で割った値のいずれか小さい値の設定の下で 2 台の IO サーバを経由してアクセスを行った．

MPI-IO の計測結果を**図 7** に示す．グラフの縦棒が書き

(a) ”ストライピング”（S_S=1 MiB） (b) ”ストライピング”（S_S=16 MiB） (c) ”ノンストライピング”（S_S=256 MiB）

図 7: ES の 96 ノード上で IOR ベンチマークを用いた 384 プロセスでの MPI-IO 性能

込みおよび読み込みの性能の平均値を表し, 平均値の上下のバーが最大値および最小値を示している. S_S を変えることで性能の違いが確認でき,S_S をある程度大きく取っておくことで性能が改善できている様子が分かる. 一方, **図 8** は TP-IO を強制的に実施させた際の性能である. 強制的に TP-IO を実施したことで, プロセス間のデータ通信と CB 単位のファイル I/O が交互に行われている. 図 7 に示したデフォルト設定での MPI-IO の場合に比べ, 著しく性能が低下しており, S_S の大きさを変化させても性能に大きな変化が無い様子が伺えるが, デフォルト設定での MPI-IO の動作パターンとの違いから, TP-IO を強制することで, プロセス間の通信が行われ, ここに要する時間が全体の性能に占める割合が高くなったためではないかと考えている.

さらに, 図 7 に示した結果との比較のために, POSIX-I/O での結果を**図 9** に示す. POSIX-I/O では, 各プロセスがはじめから独立に担当データを XFER 単位でファイル I/O を行っているが, これは, 図 7 での MPI-IO のケースにおいて, 事前に I/O パターン分析のプロセス間通信を含む処理がある点を除き, ほぼ動作パターンは同じである. こちらの場合,S_S の大きさを変えても性能に大差は無く, また XFER を変化させても性能に大きな違いは見られなかった.

次に,BLK の大きさを ScaTeFS のデフォルトの chunk size の 2 倍の 512 MiB に増やして同様のベンチマーク評価を行った. MPI-IO による評価結果に関して, **図 10** および **図 11** にデフォルト設定並びに TP-IO を強制した場合の評価結果を示す. 図 7 および図 8 の結果と比較して, 相対的に性能が低下している. 全体のファイル I/O 量が倍になったことや,BLK の大きさが chunk size よりも大きかったことも関係しているのではないかと考えており, これらの点も含め性能低下の原因調査を進めているが, 利用者側にとっては, このような特性も参考にすることで, 性能改善が行える可能性があると考えられる.

4.2 「京」における性能評価結果

ES で実施したものと同様の評価を「京」の LFS でも実施した. $4\times3\times8$ の 3 次元形状で割当てた 96 台の計算ノードに, ノードあたり 4 プロセスずつ配置し, 全体で 384 プロセスにより MPI-IO で得られた性能を**図 12** に示す. S_S は 1 MiB,16 MiB 並びに BLK 長と同じ 512 MiB の 3 パターンを行った. 書き込みでは S_S が小さい方が性能が高くなる傾向があり, 一方で, 読み込みでは S_S が大きい方がより高い性能を得やすいことが分かる.

書き込みについては S_S が小さい方が同時に動作するアグリゲータの数が多くなるためである. S_S が大きくなるにつれ, その数が減ることによって性能が低下してしまう. ただし,XFER の増大がアグリゲータ数を増やす方向に働き, 性能を向上させることができることも図 12(b) から伺える. 一方, 読み込みについては XFER が 512 MiB の時の性能低下が著しいが, これは MPI-IO 実装内での読み込み性能が低下していたことを確認している. 先読みの効果が影響していると考えられるが,「京」では FEFS からの読み込み時のクライアントキャッシュの大きさが 128 MiB になっており, 目安として一度に読み込むサイズがこのサイズを越えないようにしてファイル I/O を行うことを推奨している. やみくもに XFER を大きくし過ぎることで性能低下を招く恐れがあることがこのデータからも確認できたと考えている.

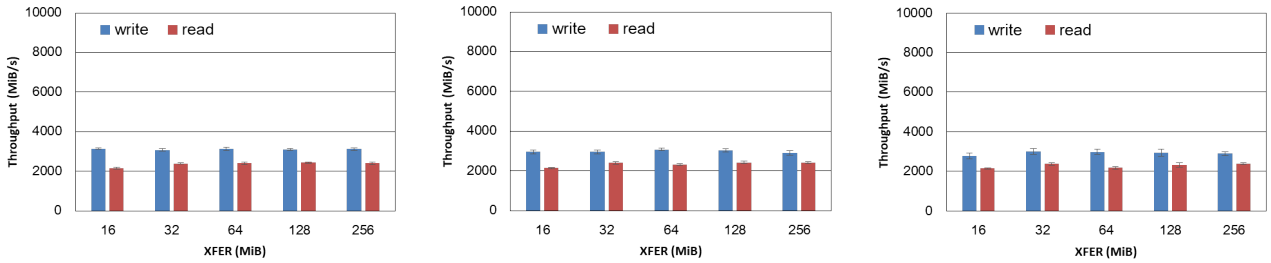
次に,MPI-IO において TP-IO を強制的に実施させた際の評価結果を**図 13** に示す. ここでは, 強制的に 16 MiB の CB 単位にファイル I/O とプロセス間通信が交互に行われており, 図 12 と比べても, XFER の大きさにほとんど関係なく同じ性能が得られている. 問題となるのは, この図に示したものでは図 13(b) のケースで, 図 12(b) と比べ, 常に 16 MiB 単位でのアクセスになるために, 同じタイミングで動作するアグリゲータの数が少なくなることで性能が低下してしまっている.

次に, 同じパラメタ設定で POSIX-I/O による性能評価を行った結果を**図 14** に示す. 図 12 と比較すると, いずれも各プロセスが独立に XFER 単位にファイル I/O を行う点で共通することもあり, XFER が 512 MiB の時に読み込み性能が著しく低下している. これは前述の通り, 読み込み時のクライアントキャッシュのサイズの影響を受けているものと考えられる. また S_S が 1 MiB の時には全般的に書き込み性能が高く, 逆に読み出し性能は S_S が 512 MiB と

© 2018 Information Processing Society of Japan

(a) ”ストライピング”（S_S=1 MiB） (b) ”ストライピング”（S_S=16 MiB） (c) ”ノンストライピング”（S_S=256 MiB）

図 8: ES の 96 ノード上に起動した 384 プロセスによる MPI-IO で TP-IO を強制した際の IOR ベンチマークでの性能（CB の大きさ=4 MiB）



(a) ”ストライピング”（S_S=1 MiB） (b) ”ストライピング”（S_S=16 MiB） (c) ”ノンストライピング”（S_S=256 MiB）

図 9: ES の 96 ノードに起動した 384 プロセスによる IOR ベンチマークを用いた POSIX-I/O の性能

大きい時に XFER の大きさによっては最も高い性能を発揮している.

なお, 直接 POSIX-I/O 関数を呼び出すのに比べて,MPI-IO 実装で I/O パターン分析により, 各プロセスが独立にファイル I/O を行う場合の性能は,I/O パターン分析等を行うためのプロセス間通信などが余分に含まれる分, 性能が低くなる.

4.3 ES と「京」での計測結果比較と MPI-IO 実装の高度化に向けた提案

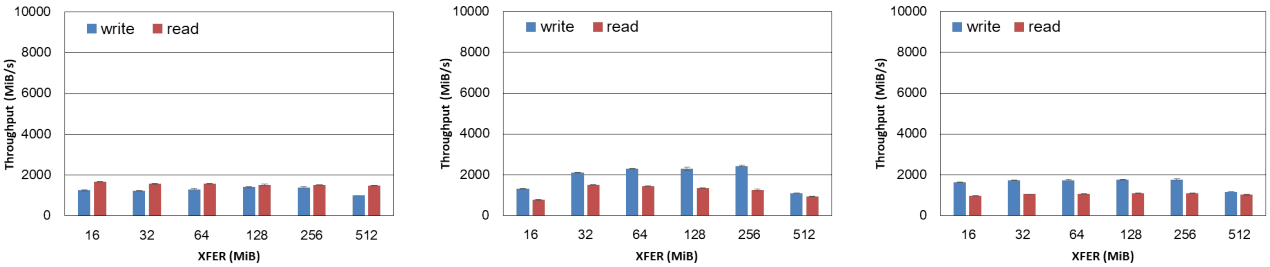
ES の ScaTeFS と「京」の FEFS は, 共に PFS ではあるが, 前述の通り, ストライピング処理機構や計算ノード群との接続方法等に違いがあり, また機器構成も異なることから, 評価データにもそれぞれの特性が現れていた. ES の場合には, 原因究明に至っていないが, 計算ノード群と work 領域間のネットワークがボトルネックになっている可能性を疑っており, ここが解消された場合には, また異なった振舞いが見られる可能性がある. 「京」においては特にストライピング処理と I/O パターンとのマッチングで性能に違いが出やすい傾向にあり, ファイル I/O 性能向上のためには, それぞれに合わせた最適化が必要となる. その意味で, 今回行った計測データも含め, センタ間での情報共有と性能情報等の開示は有用であると考えている.

また, 今回の取り組みの中で, 改めて MPI-IO 実装の高度化の必要性を再確認した. 双方ともに, 汎用的なファイルビュー指向の TP-IO 実装を用いているが, ストライピング機構を有効に活用することで性能向上が可能な PFS においては, Lustre 向けの実装のようにストライピングレイアウト指向の TP-IO をベースに実装する方が望ましいと考えている. 「京」においては,ROMIO の独自拡張実装である EARTH on K によって, システム標準のものよりも高い性能が達成されている. ストライピングレイアウト指向にすることで, 各 OST に同時にアクセスするクライアント側のプロセス数を減らすことが可能になり, ロック競合や通信の混雑による性能低下を軽減できる. さらに, アグリゲータ配置に配慮した最適化や各 OST へのファイル I/O での Throttling 機構なども性能向上に大きく貢献している. ES においても, 通信やファイル I/O の混雑等がボトルネックの要因であれば, 同様の取り組みを行うことで MPI-IO での更なる性能向上の可能性があるものと考えている.

5. 関連研究

異なるスパコン間で性能評価や分析等を行う試みはこれまでにもなされているが [19–21], それらの多くはシステムの評価あるいは運用実績の評価に主眼が置かれており, ユーザの利便性に着目して性能評価等のデータを共有する試みは少ない. それぞれのスパコンのファイルシステムの特性や, いくつかの特徴的な I/O パターンと性能の関係などの情報を提供する方が, ユーザにとってはより有用であると考えられるが, そのような情報開示は現時点において十分になされているとは言い難い. その観点で, 本稿で提案するような取り組みは今後重要になると考えている.

© 2018 Information Processing Society of Japan



(a) ”ストライピング”（S_S=1 MiB） (b) ”ストライピング”（S_S=16 MiB） (c) ”ノンストライピング”（S_S=256 MiB）

図 10: ES の 96 ノードに起動した 384 プロセスによる IOR ベンチマークを用いた BLK=512MiB での MPI-IO の性能

(a) ”ストライピング”（S_S=1 MiB） (b) ”ストライピング”（S_S=16 MiB） (c) ”ノンストライピング”（S_S=256 MiB）

図 11: ES の 96 ノードに起動した 384 プロセスによる IOR ベンチマークを用いた BLK=512MiB での TP-IO を強制した際の MPI-IO の性能

ファイルシステムやファイル I/O を評価する数々のベンチマークが存在するが, それぞれのスパコンセンターで選択するベンチマークが異なったり, あるいは同じベンチマークでもパラメタが異なるなど, 統一的に比較検討をする枠組みはこれまでには整備されていなかった. そのような中, 統一した枠組みでファイルシステムの評価とランキング公表を行う IO-500 [22] が 2017 年の SC から始まった. IO-500 では, 既存のベンチマークである IOR や MDTEST に加え, `find` によるファイル探索性能などの新しい評価コードも整備し, 総合的な評価指標による性能ランキングを行っている. 統一した評価方法によりシステム間の比較が容易になったが, 指標値は順位付けの色合いが強く, ユーザのプログラムにおける改善等に資するような情報にはなっていないことや,Burst Buffer やストレージ機器などのハードウェア構成が測定結果に大きく影響するために単純な性能比較は難しく, 必ずしもユーザにとって有用な情報にはなっていない. 開始からまだ時間が浅く, 評価方法が今後も変更されてゆく可能性もあり, 広く認識された指標値としての利用にはもう少し時間を要すると思われる. その意味では, 我々が行ったような特徴的な I/O パターン毎の性能評価結果を公開する方が, ユーザにとってより直接的で活用しやすいものと考えられる.

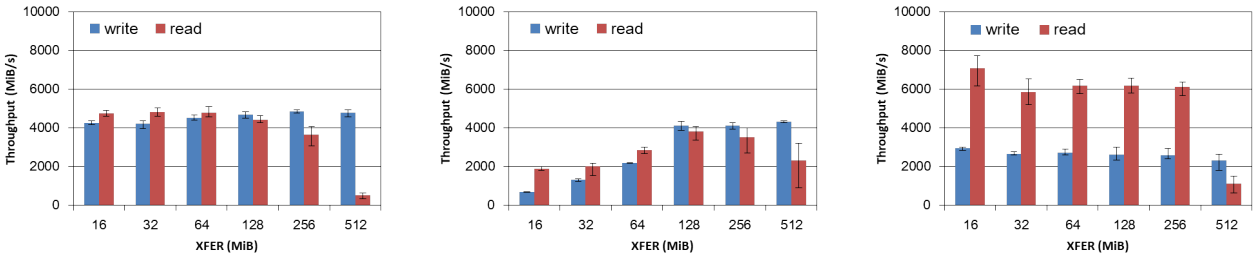
6. 本稿のまとめ

ユーザに資するような情報提供を行う枠組み作りを目標に, 異なるスパコンの PFS の性能比較や挙動等をスパコンセンター間で共有する取り組みとして, 本稿では,ES および「京」において IOR ベンチマークによる性能評価と比較検討を行った. 2 つのスパコンセンター間で様々なシステム情報や性能情報等を共有および情報提供を行いながら性能評価を進める中で,MPI-IO 実装においてベンダ毎に微妙なパラメタ名の違いなどがあることを再確認した. また, 双方の MPI-IO 実装と IOR ベンチでの I/O パターンとのマッチングを検証してゆく中で, それぞれの MPI-IO 実装や PFS の現状を把握し, さらなる性能改善について検討を行う機会の創出にも繋がった. さらに, 以上のような取り組みを通して, 各スパコンの特性や性能情報を公開する動機も与えることになり, 使用するスパコンの選択やアプリケーションでのファイル I/O の最適化などにおいて, ユーザに有用な情報提供が行える可能性を確認した.

今後, システムの高度化・複雑化がさらに進むと共に, ユーザのニーズも多様化してくると思われる. その中で, 一つのスパコンセンター内で運用技術の高度化を進めてゆくことは, 益々困難になると思われる. その意味においては, 可能な限りスパコンセンター間で情報共有を進めると共に, それぞれの運用技術およびサービスの質向上に努めるのが効果的と考えている. 今後, 情報共有を行うスパコンセンター数の拡大や, PFS 以外も対象とする情報共有の範囲拡大を検討する予定である. なお, 今回実施した性能評価結果等はそれぞれのスパコンセンターにおいて順次, 情報公開を行う予定である.

謝辞 本稿の測定結果は, 地球シミュレータならびにスー

© 2018 Information Processing Society of Japan



(a) S_S=1 MiB　(b) S_S=16 MiB　(c) S_S=512 MiB

図 12: 「京」の 96 ノード（形状：4×3×8）に 384 プロセスを起動した際の IOR ベンチマークによる MPI-IO 性能（C_S=24）

(a) S_S=1 MiB　(b) S_S=16 MiB　(c) S_S=512 MiB

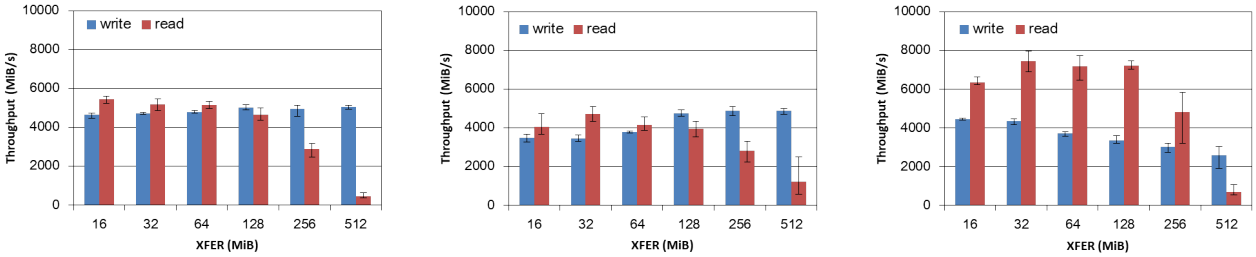
図 13: 「京」の 96 ノード（形状：4 × 3 × 8）に 384 プロセスを起動し, TP-IO を強制した際の IOR ベンチマークによる MPI-IO 性能（C_S=24, CB の大きさ=16 MiB）

パーコンピュータ「京」を用いて得られました. この場を借りて, ご支援を頂きました双方の関係者に感謝を申し上げます.

参考文献

[1] Message Passing Interface Forum: *MPI-2: Extensions to the Message-Passing Interface* (1997).
[2] MPI Forum: `http://www.mpi-forum.org/`.
[3] 辻田祐一，古谷吉隆，肥田　元，山本啓二，宇野篤也，末安史親：「京」のファイルシステムの I/O 性能改善に対する取り組み，情報処理学会研究報告，Vol. 2-17-HPC-162, No. 12, pp. 1–8 (2017).
[4] Parallel netCDF: `http://cucis.ece.northwestern.edu/projects/PnetCDF/`.
[5] The National Center for Supercomputing Applications: `https://www.hdfgroup.org/`.
[6] 国立研究開発法人海洋研究開発機構：地球シミュレータ, `https://www.jamstec.go.jp/es/jp/`.
[7] : 特集：スーパーコンピュータ「京」, 情報処理, Vol. 53, No. 8, pp. 752–807 (2012).
[8] Sakai, K., Sumimoto, S. and Kurokawa, M.: High-Performance and Highly Reliable File System for the K computer, *Fujitsu Sci. Tech. J.*, Vol. 48, No. 3, pp. 302–309 (2012).
[9] Lustre: `http://lustre.org/`.
[10] Ajima, Y., Inoue, T., Hiramoto, S., Takagi, Y. and Shimizu, T.: The Tofu Interconnect, *IEEE Micro*, Vol. 32, No. 1, pp. 21–31 (2012).
[11] 宇野篤也，庄司文由，横川三津夫：ファイルステージングのあるジョブスケジューリングの評価，情報処理学会研究報告，Vol. 2012-HPC-136, No. 22 (2012).
[12] 山本啓二，宇野篤也，塚本俊之，菅田勝文，庄司文由：スーパーコンピュータ「京」の運用状況，情報処理，Vol. 55, No. 8, pp. 786–793 (2014).
[13] Thakur, R., Gropp, W. and Lusk, E.: On Implementing MPI-IO Portably and with High Performance, *Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems*, pp. 23–32 (1999).
[14] MPICH: `http://www.mpich.org/`.
[15] Open MPI: Open Source High Performance Computing, `http://www.open-mpi.org/`.
[16] Thakur, R., Gropp, W. and Lusk, E.: Optimizing noncontiguous accesses in MPI-IO, *Parallel Computing*, Vol. 28, No. 1, pp. 83–105 (2002).
[17] Lustre: Lustre ADIO collective write driver, Technical report, Lustre (2008).
[18] Tsujita, Y., Hori, A., Kameyama, T., Uno, A., Shoji, F. and Ishikawa, Y.: Improving Collective MPI-IO Using Topology-Aware Stepwise Data Aggregation with I/O Throttling, *Proceedings of HPC Asia 2018: International Conference on High Performance Computing in Asia-Pacific Region, January 28-31, 2018*, ACM, pp. 12–23 (2018).
[19] Behzad, B., Luu, H. V. T., Huchette, J., Byna, S., Prabhat, Aydt, R., Koziol, Q. and Snir, M.: Taming Parallel I/O Complexity with Auto-Tuning, *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, ACM, pp. 68:1–68:12 (2013).
[20] Luu, H., Winslett, M., Gropp, W., Ross, R., Carns, P., Harms, K., Prabhat, M., Byna, S. and Yao, Y.: A Multiplatform Study of I/O Behavior on Petascale Supercomputers, *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, ACM, pp. 33–44 (2015).
[21] Domke, J., Hoefler, T. and Matsuoka, S.: Fail-in-Place Network Design: Interaction Between Topology, Routing Algorithm and Failures, *Proceedings of the Inter-*

© 2018 Information Processing Society of Japan



(a) S_S=1 MiB　(b) S_S=16 MiB　(c) S_S=512 MiB

図 14: 「京」の 96 ノード（形状：4 × 3 × 8）に 384 プロセスを起動した際の IOR ベンチマークによる POSIX-I/O 性能（C_S=24）

national Conference for High Performance Computing, Networking, Storage and Analysis, SC'14, IEEE Press, pp. 597–608 (2014).

[22] IO-500: `https://www.vi4io.org/std/io500/start`.

© 2018 Information Processing Society of Japan