Vol.2019-HPC-169 No.6
2019/5/10

「京」におけるファイルシステムとI/Oノードのログ情報を用いたファイルI/Oの最適化支援の取り組み

辻田 祐一[1,a)]　古谷 吉隆[2]　肥田 元[3]　宇野 篤也[1]

概要：スーパーコンピュータ「京」（以下，「京」）の安定運用と高度利用に向けた重要な取り組みの一つとして, 我々はファイルシステムの運用改善を進めている．「京」では Lustre をベースとしたファイルシステムを採用しており，8 万台を超える計算ノードが 1 つのファイルシステムに接続されている．計算ジョブによってはファイルシステムへのアクセス効率が実行時間に与える影響が大きいものもあり, ファイル I/O の最適化が重要なケースも散見されている．「京」においては, 計算ジョブの最適化に資するツール群がユーザ向けに提供されているが, それらは計算処理やノード間通信の最適化に対するものであり, ファイルシステムや I/O ノードにおけるファイル I/O の最適化に関するものは現時点で提供されていない．一方, 「京」の運用支援の一環として, ファイル I/O に関係するファイルシステム内部の統計情報や I/O ノードにおける通信状況などのログ情報を収集しており, 計算ジョブにおけるファイル I/O の最適化に向けて, これらの情報の活用の可能性について検証を進めている．本稿では, ファイル I/O として集団型 MPI-IO の高速化を目指した実装である EARTH on K を一事例として, 本取り組みの有効性について検証を行った．この検証から, ファイルシステム内の統計情報や I/O ノードにおける通信の混雑状況の分析がファイル I/O の最適化の検証に有用であることを確認した．

キーワード：スーパーコンピュータ「京」，Lustre，FEFS, I/O ノード，OSS，Tofu

1. はじめに

近年の HPC システムの多様化かつ大規模化によって，計算ジョブで扱うデータ量は増加の一途を辿っている．データが格納されるファイルシステムには，大容量かつ高性能であると共に, 高い安定性が求められており, GPFS [1] や Lustre [2] 等の並列ファイルシステムが広く用いられている．このような並列ファイルシステムを利用した大規模ファイル I/O において, I/O 性能が実行時間に与えるインパクトは大きいため, 計算ジョブの計算処理やノード間通信などの最適化と同様にファイル I/O の最適化が益々重要になってきている．

HPC システムの多くは, 計算処理やノード間通信の最適化に重点を置いており, これらの最適化を支援する多くのツール群 [3–5] が提供されている．一方，ファイル I/O の最適化に関しては，前者ほどは多くはないが，例えば I/O パターン分析に広く用いられている Darshan [6] などのプロファイリングツールがある．しかしながら，ファイルシステムや，ファイルシステムへのアクセス時に経由する I/O ノードの動作状況までも含めたプロファイリングはサポートしていない．最近では，ファイルシステムに関する監視や性能改善を目的としたツールとして，例えば Cray View for ClusterStor [7] と呼ばれるフレームワークが Cray の HPC システム向けに提供されており，問題箇所の早期発見などに利用されている．他には，TOKIO (Total Knowledge of I/O) [8] と呼ばれるシステム全体でファイル I/O のトレース並びに分析を支援するフレームワークが提案されており，ファイルシステム内部の振舞い等も含めた総合的な分析機能を提供する取り組みが強化されつつある．

「京」においては，インターコネクトである Tofu [9] に関する通信情報プロファイリング機能である Tofu PA [10] など，様々な最適化支援ツール群が用意されているが，ファイル I/O に関しては「京」向けに移植および機能拡張を行った Darshan [11] による I/O パターンのプロファイリングが利用できるのみで，ファイルシステムやファイル I/O の際に経由する I/O ノードに対するプロファイリング機能は提供されておらず，ファイルシステムや I/O ノードでの振舞いを確認する方法は無い．一方で，運用支援業務の取り組みの一つとしてファイルシステムの統計情報や I/O ノー

[1] 理化学研究所 計算科学研究センター
[2] 富士通株式会社
[3] （株）富士通ソーシアルサイエンスラボラトリ
[a)] yuichi.tsujita@riken.jp

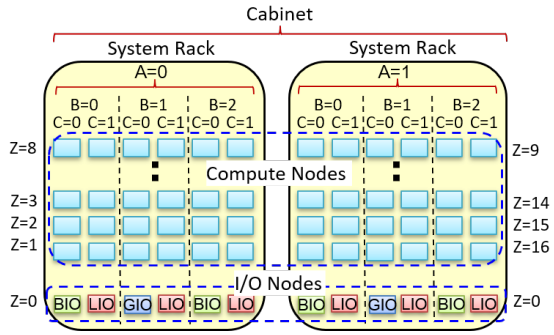


図 1: 計算ノードおよび I/O ノードのシステムラック内の配置

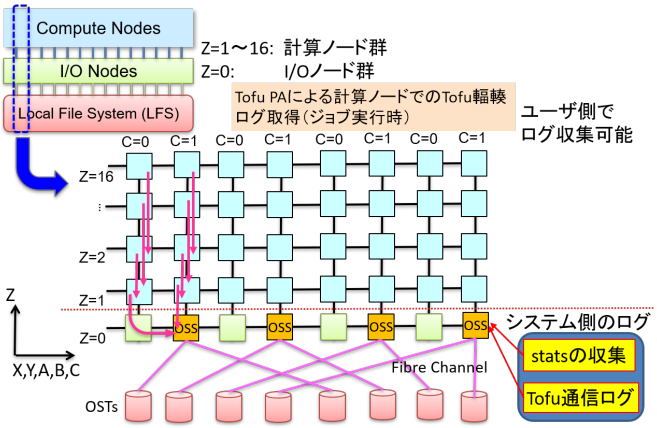


図 2: ユーザが利用可能な Tofu PA 機能とファイルシステムに関する運用側で収集しているログ情報

ドの通信状況などを定期的に監視およびログ収集を行っており，運用上の問題が発生した際などに活用している．これらのログデータは現時点でユーザ向けに公開していないが，前述の総合的な分析機能の取り組みの強化の動きなども勘案すると，ファイル I/O の最適化に有用であると考えられるため，今回，「京」における MPI-IO 拡張実装である EARTH on K (以下，EARTH) [12] を例として最適化への活用の可能性を検証したところ，ファイルシステムや I/O ノード側の振舞いから，最適化の効果などの比較に有効活用できる可能性が確認できた．

以下，第 2 章では，「京」におけるファイルシステムの概略とログ情報収集状況に関して説明し，第 3 章において，今回提案するログ情報を用いたファイル I/O 最適化支援機能について説明する．次に，EARTH を用いた提案手法の有効性の検証試験について，第 4 章で報告する．さらに，関連研究について第 5 章で述べた後に，第 6 章で本稿のまとめを行う．

2. 「京」のファイルシステムとログ収集・性能監視

本章では，「京」のファイルシステムの構成と，ファイル I/O に関連して，「京」でユーザが利用可能なプロファイリング機能である Tofu PA，並びに運用側で収集しているログ情報の現状について説明する．

「京」のファイルシステムは，データやプログラム等を保管するグローバルファイルシステム（GFS）と計算中の高速な I/O に利用されるローカルファイルシステム（LFS）により，2 階層で構成されている．各々のファイルシステムには Lustre をベースに富士通により開発された FEFS (Fujitsu Exabyte File System) [13] が用いられている．

計算ノードおよび I/O ノードは**図 1** に示す配置で 96 台の計算ノードおよび 6 台の I/O ノードが 1 つのシステムラック内に収められており，2 つのシステムラックで 1 つのキャビネットを構成している．1 つのキャビネットが 6 次元で記述される配置（X, Y, Z, A, B, C）において，X 座標および Y 座標が同一となっている．同一キャビネット内の各々のシステムラックの座標 A は 0 または 1 となっており，同一システムラック内は B 座標および C 座標の組み合わせで 6 つのグループに分かれている．システムラック間において，同一の B 座標および C 座標の計算ノード 16 台と 1 台の I/O ノードで一つのトーラス結合の Z 軸を形成する．I/O ノードには BIO，LIO および GIO の 3 種類があり，図に示す配置で各システムラック内に収められている．なお，LFS の Object Storage Server(OSS) は LIO 上で動作している．I/O ノード・計算ノード間は Tofu で結合されており，各ノードには 10 方向の通信経路（X+, X-, Y+, Y-, Z+, Z-, A, B+, B-, C）がある．

LFS における各計算ノードからのファイル I/O において，ユーザが利用可能な Tofu PA と運用側で収集しているログ情報の概略を**図 2** に示す．この図に示すように，OSS が動いている各 LIO 配下には，2 台の Object Storage Target（OST）が Fibre Channel で接続されている．計算ノードからのファイル I/O のみ，同一 Z 軸上の I/O ノードを含め Z 軸がトーラス結合になり（Z=16 の計算ノードが当該 I/O ノードと接続），当該ノードから近い方向で当該 I/O ノードを経由して，対象の OST へ LIO 上で動作する OSS を通してアクセスする．よって，計算ノードからの OST 群へのアクセスにおいて，同一 Z 軸上の計算ノード間だけでなく，I/O ノード間でも Tofu 上のデータ通信が発生する．

ユーザのジョブが動作した計算ノード群での Tofu における通信のプロファイリングを支援するツールが Tofu PA であり，当該ツールによって，使用した計算ノード毎に Tofu の各方向の通信状況がプログラム実行時に取得できる．取得できる通信情報としては，大きく分けて，データ転送サイズと通信時の相手先への送信完了までに要したサイクル数の 2 種類があり，後者は通信混雑状況の指標としても利用できる．しかしながら，ファイルシステムや I/O ノード側の振舞いをプロファイリングする機能はユーザ

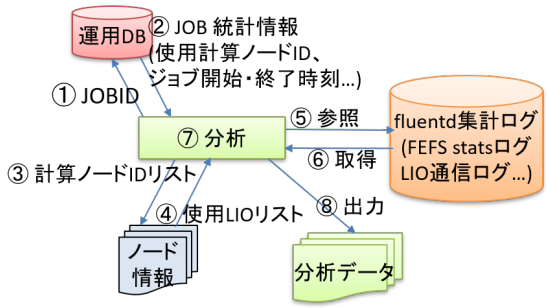


図 3: ファイルシステムおよび I/O ノードのログ情報を用いたファイル I/O 最適化支援機能

に提供されておらず，現状では，ファイルシステムや I/O ノードのプロファイリング情報を用いたファイル I/O の最適化は行えない．

一方，「京」の運用においては，数々の運転状況の監視の取り組みの一つとして，OSS の stats 情報や LIO を含む I/O ノード群における Tofu 上の通信情報を fluentd [14] によって定期的にログ収集並びに監視を行っている．現時点では，stats 情報の収集は 1 分間隔，I/O ノードの Tofu 通信情報の収集は 10 分間隔で行っている．収集されたログ情報は運用側でのファイルシステムの高負荷分析等に限定しており，ユーザには公開されていない．

3. ログ情報を用いたファイル I/O 最適化支援機能

「京」におけるファイル I/O の最適化における問題に鑑み，ファイルシステムおよび I/O ノードから収集している前述のログ情報に関して，ファイル I/O の最適化などに利活用できる可能性があると考え，我々は，**図 3** に示す最適化支援機能を提案する．提案する機能において，「京」におけるジョブの統計情報等を格納する PostgreSQL データベース（図中の”運用 DB”）に対し，ジョブ ID を検索キーとして（図中の 1），当該ジョブの統計情報を入手する（図中の 2）．入手した情報の中にある，使用された計算ノード群のノード ID から（図中の 3），ノード情報のテーブル (図中の”ノード情報”) を用いて使用された LIO ノード群の情報を入手する（図中の 4）．

その後，ここまでで得られた使用した LIO ノード群や当該ジョブの開始・終了時刻などを用い（図中の 5），fluentd で収集されているログ群の中から，ファイルシステムの OSS 群の stats 情報や LIO での Tofu 上の通信ログを取得する（図中の 6）．得られた stats 情報や通信ログから分析・データ処理を行った後に（図中の 7），使用した LIO で動作する OSS に関する CSV 形式のプロファイリング情報や，LIO での Tofu 上の通信の混雑状況のヒートマップ図が出力される（図中の 8）．

FEFS の OSS における stats（`/porc/fs/lustre/ost/OSS/ost/stats`）内には，数々のパラメタ類があるが，本提案手法では試験的に**表 1** に示す 3 種類のパラメタを用いた．なお，stats 情報に関しては，Lustre における stats 情報の活用方法に倣い，取得間隔での平均値および毎秒の処理数を，提案する支援機能内部において算出している．

表 1: 提案手法で用いた OSS の stats 内のパラメタ一覧

パラメタ名	当該パラメタの内容
snapshot_time	パラメタ値を採取した UNIX 時間
req_waittime	キュー内に格納されてからキューから取り出され処理が開始されるまでの待ち時間
req_qdepth	処理開始を待っているキュー内に格納されているリクエストの数
req_active	パラメタ取得時点において処理が進行中のリクエスト数

4. ファイル I/O の最適化支援機能の活用事例

提案するファイル I/O 最適化支援機能の活用事例として，EARTH の最適化検証への利用例を以下説明する．

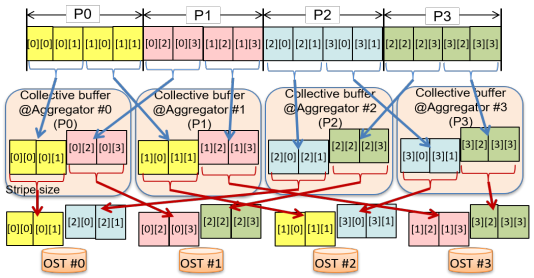
4.1 EARTH on K

集団型 MPI-IO の実装内部において Two-Phase I/O [15]（以下，TP-IO）と呼ばれる最適化手法が用いられている．TP-IO では，各々のプロセスが自身のアクセス対象とは関係無く，ファイルビュー全体をプロセス間で分割して，できる限り連続なファイル I/O を行う．そのためにプロセス間でファイル上のデータの並びに合わせたデータ通信と前述のファイル I/O を組み合わせ，有限のバッファサイズ単位でファイル全体のアクセスが終了するまで処理を繰り返す．

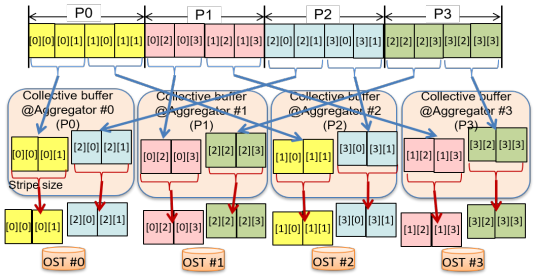
ここで，TP-IO に関して「京」の標準 MPI ライブラリが提供する MPI-IO 機能（以下，標準 MPI-IO）で判明していた通信やファイル I/O の混雑の問題を解消するために EARTH が提案されている．EARTH は「京」の集団型 MPI-IO 高速化の実現に向けて上述の TP-IO に対して数々の最適化を施しており，標準 MPI-IO と比較して以下のような特徴を持っている．

(1) ストライピングアクセス指向の TP-IO 実装
(2) ファイル I/O を行うプロセス（アグリゲータ）のストライピングアクセスに最適化された割付け配置
(3) ファイルシステムへのアクセス性能向上を目指した I/O 要求の段階的発行 (I/O Throttling) および I/O Throttling に合わせた段階的なプロセス間データ通信

最初の項目に関して，標準 MPI-IO の TP-IO では，ファイルビューでアグリゲータ間で均等にアクセス領域を分割しているが，**図 4**(a) に示すように，各アグリゲータが多数の OST へアクセスするため，OST へのアクセスの混

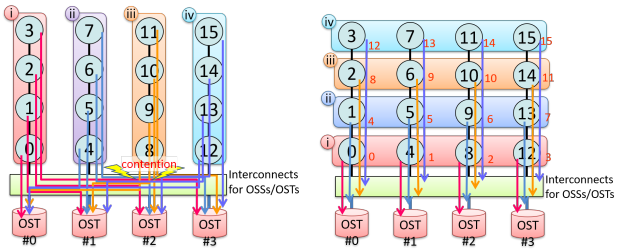


(a) ファイルビュー指向



(b) ストライピングアクセス指向

図 4: ファイルビュー指向およびストライピングアクセス指向の TP-IO



(a) 標準 MPI-IO　　(b) EARTH

図 5: 標準 MPI-IO および EARTH におけるアグリゲータ割付け配置

雑による性能低下の可能性がある．一方，EARTH では，図 4(b) に示すストライピングアクセス指向の方式を採用している．この方式は，現在の Lustre 向け実装 [16] にも採用されている方式であり，ファイルビュー指向で問題のあったアクセス混雑を軽減させて性能向上を実現している．

次に，2 番目の項目に関しては，**図 5** に示すように，EARTH においては，アグリゲータの割付け配置の改善によりファイル I/O における混雑緩和を実現している．この図に示す丸囲みの数字はプロセスを表しており，数字は rank である．標準 MPI-IO においては，アグリゲータ割付の順番がランク順になっており，性能がランク配置に大きく影響されると共に，ストライピングアクセスに向けた最適化がなされていないために，この図のように多数のアグリゲータからのファイル I/O において競合・混雑が発生し，十分な性能が得られない．よって，ファイル I/O 時に I/O リクエストの混雑が I/O ノードあるいはファイルシステム側で発生していることが予見される．一方，EARTH はこのような問題を改善するために，ランク配置に関係なく配下のファイルシステムへのストライピングアクセスを考慮した最適なアグリゲータ配置を実現しており，この図においては，丸囲みの数字の右側にある数字の順で割り当てられる．この最適化により，標準 MPI-IO と比較して，I/O リクエストの混雑緩和が実現されることが期待される．

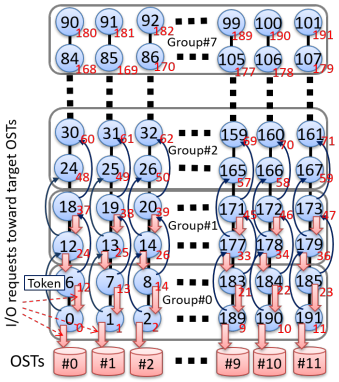


図 6: EARTH における I/O Throttling

最後に 3 番目の項目に関しては，**図 6** に示すように，各プロセスから発行される I/O 要求をトークンリレー方式によって，段階的に発行数を絞る I/O Throttling を採用したことで，ファイルシステムの過負荷・混雑を軽減させ，ファイル I/O 性能を向上させている．なお，図中の丸囲みの数字は図 5 と同様にプロセスを表し，数字は rank である．また各丸囲みの数字の右下にある数字はアグリゲータ割付けの順番である．ランク順とは関係なく，アグリゲータの割付け順と各 OST に対して同時に発行するリクエスト数（図 6 では 2 個ずつのパターン）に応じてグループ分けを行い，トークンの受け渡しを段階的に行う．また，I/O Throttling の段階的処理に合わせる形で，プロセス間の全対全のデータ通信も段階的に進めることで，プロセス間の通信時間の低減も実現している．これらの最適化によって標準 MPI-IO と比較して，ファイルシステムや I/O ノードにおける I/O 要求の混雑緩和や計算ノード間のデータ通信の混雑緩和が実現されていると考えられる．

以上のような EARTH における最適化実装を進めるにあたり，計算ノード側で得られる Tofu PA を含むツール群等で得られる情報も活用して性能改善の検証・確認をしてきたが，ファイルシステムや I/O ノードに関しては検証手段が無かったために，この部分において EARTH における最適化の効果が確認できていなかった．そこで，今回の提案手法を用いて，ファイルシステムや I/O ノード側の振舞いを確認した．

4.2　ベンチマークによる性能評価における活用事例

EARTH を用いて評価するにあたり，HPIO ベンチマーク [17] を用い，**図 7** に示すように region size, region space,

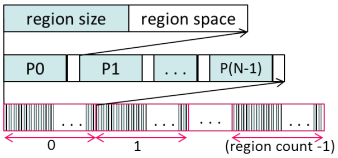


図 7: HPIO ベンチマークにおけるアクセスパターン．$P0$ から $P(N-1)$ で示される領域がランクが 0 から N-1 までのプロセスのアクセス対象

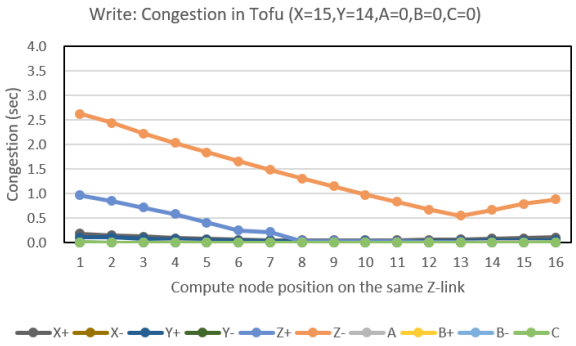
表 2: HPIO ベンチマークによる I/O 性能の平均値

	標準 MPI-IO	EARTH
書き込み	18.2 GiB/s	25.3 GiB/s
読み出し	38.9 GiB/s	45.6 GiB/s

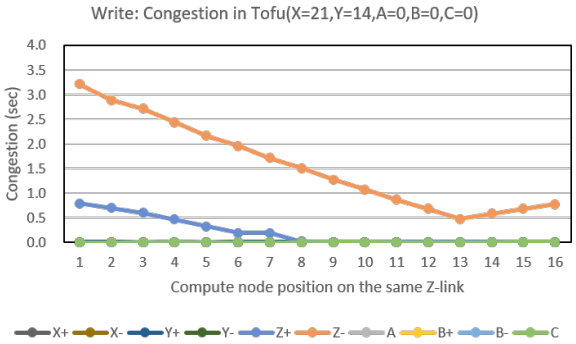
及び region count の 3 種類のパラメタを，それぞれ 5,992 バイト，256 バイト，並びに 30,729 に設定し，飛び飛びのアクセスパターンを MPI の派生データ型関数により生成し，これを用いた集団型 MPI-IO による書き込み処理，並びにこれに続く読み出し処理の評価を実施した．標準 MPI-IO を評価の比較対象とし，3,072 台の計算ノードを $8 \times 12 \times 32$ の 3 次元の形状指定で確保し，ノードあたり 4 プロセスを起動させて，12,288 プロセスにより上記の条件の下で評価を行った．なお，上記の形状指定により，96 台の LIO が割り当てられ，それらの配下に合計で 192 台の OST が割り当てられた．ファイルシステム側の設定に関しては，確保した計算ノード群で使用できる最大のストライプカウントである 192 を設定し，ストライプサイズは 256 MiB とした．

まず初めに標準 MPI-IO および EARTH によるベンチマーク結果を**表 2** に示す．EARTH による最適化によって書き込みおよび読み出し共に標準 MPI-IO と比較して性能が改善されていることが分かる．性能改善は実装内部のプロセス間のデータ通信やファイルシステムへの I/O に適用した最適化によるものであるが，ユーザ向けに提供されている Tofu PA で得られる計算ノードでの通信状況からは検証が十分に行えなかった経緯がある．そこで，今回提案するファイル I/O 最適化支援機能を用いることで，最適化の効果の検証が可能になった経緯を，以下，Tofu PA での検証も含めて各検証項目毎に報告する．

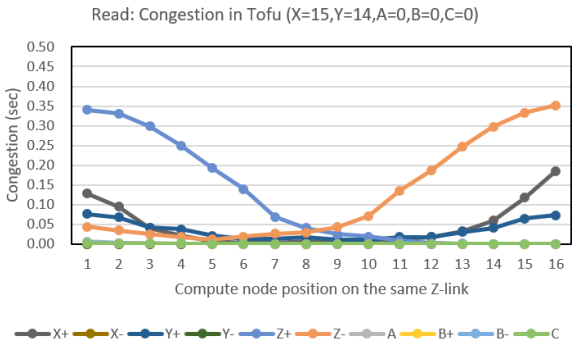
- Tofu PA による計算ノードにおける通信状況の確認
  まずはじめに Tofu PA により，集団型書き込みおよび読み出し関数における計算ノードでの通信混雑状況を確認した．使用した 3,072 ノードのうち，rank=0 が配置された計算ノードを含む同一 Z 軸上の計算ノード群で通信が行われた Tofu の各軸に対し，書き込みおよび読み出しの処理を各々 6 回実施した中から最大値を求めたものを**図 8** に示す．各グラフの横軸は測定した計算ノードの Z 軸上の座標である．書き込みにおいては，標準 MPI-IO と比較して，EARTH で特徴的な違いは見られなかった．一方，読み出しにおいては，標準 MPI-IO と比べ，EARTH の方が混雑状況が



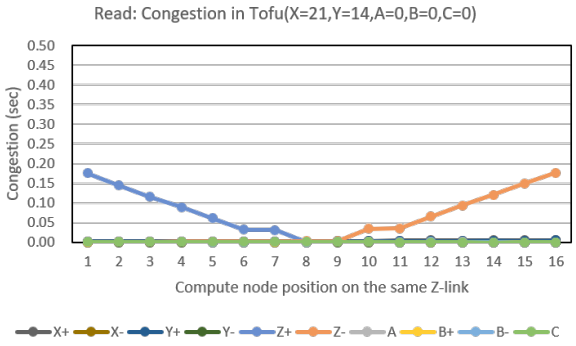
(a) 標準 MPI-IO（書き込み）



(b) EARTH（書き込み）



(c) 標準 MPI-IO（読み出し）



(d) EARTH（読み出し）

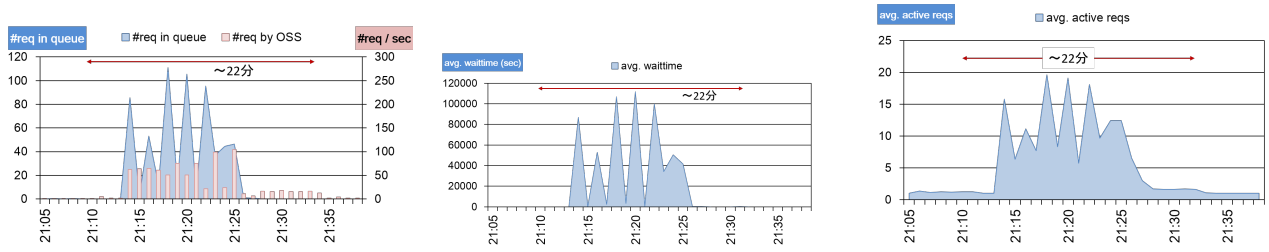
図 8: Tofu PA による各通信軸の通信混雑状況

若干改善されており，標準 MPI-IO では Z 軸の 2 方向（Z+および Z-）以外にも．X 軸や Y 軸にもそれなりの混雑が確認でき，MPI-IO 実装内部のプロセス間データ通信に関して，EARTH の方が標準 MPI-IO よりも改善されている可能性がある．しかしながら，読み出し処理時間に占める通信混雑時間は性能に影響が出るほど長くなく，書き込みにおいては，標準 MPI-IO と EARTH とで有意な差は見られないことから，これらの情報だけで EARTH における最適化実装の効果を検証するのは難しかった．

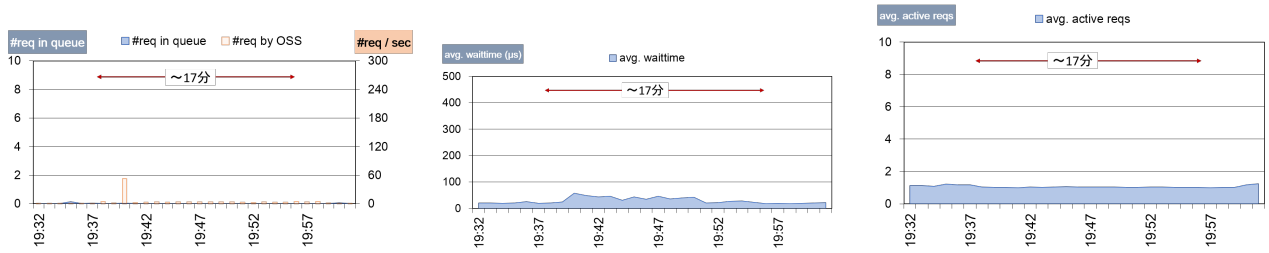
- 提案手法による OSS の stats 情報の分析
  次に，提案手法により「京」の標準 MPI-IO 並びに EARTH に対して HPIO ベンチマークにより集団型 MPI-IO による書き込み及び読み出しの処理中に使用された LIO 上で動作する OSS の stats から，`req_qdepth`，`req_waittime`，`req_active` の 3 種類の状況を調べた．なお，HPIO ベンチによる評価では，集団型 MPI-IO 関数を実行していることから，全プロセスに均等に I/O 処理が分散され，LIO 間の処理も均等になるため，使用した LIO 群の中から 1 ノードの振舞いについて確認を行った．
  標準 MPI-IO および EARTH における状況を確認したものを**図 9** に示す．`req_qdepth` においては 1 分間のサンプリング間隔毎のリクエストキュー内に滞留する平均的なリクエスト数と，毎秒の I/O リクエストの処理数を示しており，`req_waittime` および `req_active` のグラフにおいては，それぞれ 1 分間のサンプリング間隔毎の処理されるまでのキュー内での平均待ち時間並びに平均的な同時リクエスト処理数を示している．なお，図中の横線で示した区間がベンチマークが動作していた時間帯であり，標準 MPI-IO では約 22 分，EARTH では約 17 分であった．
  EARTH の場合と比較して標準 MPI-IO ではいずれのパラメタにおいても高めの値を呈していた．例えば `req_qdepth` においては，標準 MPI-IO では I/O 処理中に数十リクエストが滞留している様子が伺えるが，一方，EARTH においては，リクエストの滞留が見られない．`req_waittime` について標準 MPI-IO では最高で 100 ミリ秒を越える時間が確認出来たが，EARTH での同パラメタは長くても 50 マイクロ秒程度とリクエストのキュー内での待ち時間が極力短い．これらのことから，EARTH による最適化がファイルシステムの LIO における混雑緩和に寄与していることが確認できる．また，`req_active` において標準 MPI-IO では，多いところで 15 個から 20 個のリクエストが同時実行されているのに対し，EARTH では平均的に 1 個近辺を推移しており，LIO 上の OSS へ加わる負荷が EARTH の方が小さいことも分かる．この違いは，標準 MPI-IO におけるファイルビュー指向の TP-IO によって，各 OST が多数のアグリゲータからアクセスされるのに対し，EARTH では，ストライピングアクセス指向の TP-IO によって，各 OST 毎にアクセスするアグリゲータが固定化されていたことによると考えられる．
- 提案手法による I/O ノードの Tofu 上の通信状況の確認
  最後に各々のジョブ実行で割り当てられたローカル I/O 用の LIO 群における Tofu 通信の混雑状況を**図 10** に示す．横方向が X 軸方向の並び，縦方向が上からの向きで Y 軸方向での並びになっており，各 LIO に対応するブロックの色が濃いほど混雑が激しいことを意味する．I/O ノードでの 10 分毎の集計データの中で得られた各通信方向の混雑指標となる個々のリクエストが完了するまでに要した時間の中から最大値を LIO 毎に集計したものになっており，これまでの運用実績から，通信リクエストの完了までに要した時間が 60 秒以上が輻輳状態と判断し，ヒートマップの最大値を 60 秒として表示している．
  このヒートマップから標準 MPI-IO においては通信混雑は発生していないものの、通信状況はさほど多くなかったものと考察される．前述の OSS における stats 情報の分析では I/O リクエストの滞留が EARTH と比較して顕在化していたことなどからも，ファイルシステム側で I/O リクエストの処理が行われる以前のレベルにボトルネックがあり，OSS が動作する LIO を含めた I/O ノード間のデータ通信の負荷が小さく，ファイル I/O の性能を十分に発揮できていなかったと考えられる．一方，EARTH では標準 MPI-IO と比較して通信負荷が大きかったと思われるが，極端には局在しておらず，アグリゲータ配置の最適化によって，ファイル I/O を行うプロセスをバランス良く配置したことにより，LIO およびインターコネクト全体を比較的バランス良く利用していたものと考えられる．

5. 関連研究

近年のファイル I/O の大規模化に伴い，様々なファイル I/O の最適化を支援するツールがこれまでに開発されている．Darshan [6] は，計算ジョブのファイル I/O のトレースとトレース情報の可視化を行うツールセットであり，IBM BlueGene シリーズや Cray XE6，Linux クラスタシステムなど，GPFS や Lustre を有する多種類のシステムにおいて利用可能である．なお，Darshan は，R-CCS 公開ソフトとして「京」向けにも移植されており，利用可能である [11]．また，Darshan は単体での利用だけでなく，最近では，全体的なシステム性能分析フレームワークの I/O トレースツールとしての利用も多い [8,18]．

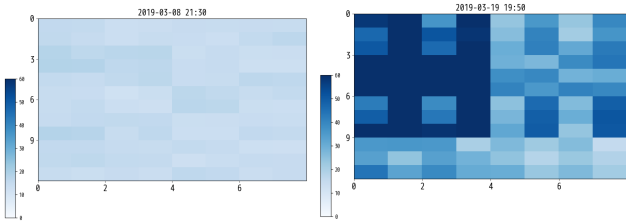


(a) 標準 MPI-IO: `req_qdepth` (b) 標準 MPI-IO: `req_waittime` (c) 標準 MPI-IO: `req_active`



(d) EARTH: `req_qdepth` (e) EARTH: `req_waittime` (f) EARTH: `req_active`

図 9: 標準 MPI-IO（(a)~(c)）および EARTH((d)~(f)) による各種 stats 情報



(a) 標準 MPI-IO (b) EARTH

図 10: 使用した LIO における Tofu 上の通信の混雑状況のヒートマップ．横軸は X 軸方向の配置順，縦軸は上から Y 軸方向での配置順．

ファイル I/O の最適化においては，ファイル I/O の自動的な最適化の取り組みを行った Watkins らの研究がある [19]．彼らの提案手法は Legion [20] と呼ばれるデータモデルを用いており，計算コード内において論理的に分割できる部分を検知し，使用する計算機やファイルシステムのシステム構成とのマッチングにより最適化を行っている．Cray が提供する HPC システムで利用できる Cray View for ClusterStor [7] は，ファイルシステム内の統計情報や性能情報の監視・分析および可視化によって，問題のある箇所の早期の特定などに寄与することを目的としている．我々がファイルシステムの運用で監視・収集しているログ情報の活用の方向性としては，上述の Cray View for ClusterStor の取り組みに近い．なお，我々の取り組みでは，運用に限定せず，ユーザプログラムにおけるファイル I/O の改善に資することも検討を進めている．

計算ノード群から並列ファイルシステムへのアクセスにおいて，高速インターコネクトが重要な役割を担うが，この通信経路の混雑状況の監視も性能改善において有用である．例えば、Cray の Gemini ネットワークでは，パフォーマンスカウンタを利用した混雑状況の監視が可能であり [21]，ファイル I/O においても同様の混雑状況のプロファイリングと I/O 性能の関連性を検証している [22]．Tofu においても同様のパフォーマンスカウンタが Tofu PA [10] と呼ばれるプロファイリングツールを通して利用できる．Tofu PA による通信プロファイリングは計算ノード群のみユーザがプログラム実行時に実施可能だが，今回，我々は「京」の運用監視の目的で収集している I/O ノードにおける Tofu 上の通信のログ情報や，I/O ノード上で動作する OSS の stats 情報を活用したファイル I/O 最適化支援機能を提案している．動作している計算機システムなどの違いはあるが，我々の取り組みは，前述の取り組み [8, 18, 22] の方向性と共通している点がある．我々の場合，さらにファイルシステム側の stats で得られる種々の統計情報を加えて分析を行っており，[8] のようなシステム全体でのログ情報分析によるプロファイリングおよび性能改善に寄与することを目指している．

6. 本稿のまとめ

「京」の運用活動において収集・分析を行っているファイルシステムと I/O ノードのログ情報を用いたユーザのジョブにおけるファイル I/O の最適化支援の可能性について検証を行った．ユーザ向けには計算処理やノード間通信のプロファイリング支援機能が利用できるが，ファイルシステムや I/O ノードの振舞いのプロファイリングが行えず，ファイル I/O の最適化に困難さが残っていた．特に大規模システムにおけるファイル I/O においては，小規模な検証プログラムでは知り得なかった問題などが顕在化する

ことが良くあるが，特にファイル I/O においてはファイルシステムや I/O ノードに対するプロファイリング機能が提供されていないため，さらに大きな困難があった．

今回，上記のログ情報の活用可能性を探るために，ジョブ ID から使用した I/O ノード上で動作する OSS の stats 情報からの分析や使用した I/O ノードにおける通信状況のログ情報を用いた通信混雑のヒートマップ作成を支援するフレームワークを試験的に実装した．また，これを用い，「京」で利用できる MPI-IO の拡張実装である EARTH を一例として，「京」の標準 MPI-IO を比較対象とした EARTH の最適化実装におけるファイルシステムや I/O ノードに与える影響についての分析等を行った．EARTH および標準 MPI-IO による HPIO ベンチマーク試験を「京」の 3,072 台の計算ノード上で 12,288 プロセスにより実施し，その際に行ったファイルシステムや I/O ノードの状況の比較検討から，これまで分からなかったファイルシステムおよび I/O ノードの振舞いに関して，それぞれの実装における振舞いの違いや EARTH における最適化実装による性能改善効果などが確認できた．

今後の課題として，今回実装した最適化支援機能を，より広く活用してゆく観点から，当該機能の使い方や活用方法などについて，今回の評価に用いた EARTH の最適化検証を継続すると共に，他のアプリケーションでのニーズなども考慮した改善を行う．これ以外にも，更なるログ情報分析の内容や質の向上に向けて必要とされるログ情報内のパラメタ群の選定や，支援機能の利便性の改善などについても今後の課題として対応を進めてゆく予定である．また，ログ情報を用いた分析データ等のユーザへの開示方法についても，今後検討を進める必要があると考えている．さらに，他のサイトのシステムでの同様の取り組みの可能性なども並行して検討を進めてゆきたいと考えている．

謝辞 本稿で使用したログ情報およびジョブ実行結果は，理化学研究所の「京」を利用して得られたものである．

参考文献

[1] Schmuck, F. and Haskin, R.: GPFS: A Shared-Disk File System for Large Computing Clusters, *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST '02, USENIX Association (2002).
[2] Lustre: `http://lustre.org/`.
[3] Scalasca: `http://www.scalasca.org/`.
[4] Vampire: `https://vampir.eu/`.
[5] Tau: `http://www.cs.uoregon.edu/research/tau/home.php`.
[6] DARSHAN: `http://www.mcs.anl.gov/research/projects/darshan/`.
[7] Langer, P.: Diagnosing Performance Issues in Cray ClusterStor Systems, *2018 Cray User Group Meeting (CUG)* (2018).
[8] Lockwood, G. K., Wright, N. J., Snyder, S., Carns, P., Brown, G. and Harms, K.: TOKIO on ClusterStor: Connecting Standard Tools to Enable Holistic I/O Performance Analysis, *2018 Cray User Group Meeting (CUG)* (2018).
[9] Ajima, Y., Inoue, T., Hiramoto, S., Takagi, Y. and Shimizu, T.: The Tofu Interconnect, *IEEE Micro*, Vol. 32, No. 1, pp. 21–31 (2012).
[10] Ida, K., Ohno, Y., Inoue, S. and kazuo Minami: Performance Profiling and Debugging on the K computer, *Fujitsu Sci. Tech. J.*, Vol. 48, No. 3, pp. 331–339 (2012).
[11] R-CCS System Software Research Team: Darshan, `https://www.sys.r-ccs.riken.jp/releasedsoftware/ksoftware/darshan/`.
[12] 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).
[13] 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).
[14] fluentd: `https://www.fluentd.org/`.
[15] Thakur, R., Gropp, W. and Lusk, E.: Optimizing noncontiguous accesses in MPI-IO, *Parallel Computing*, Vol. 28, No. 1, pp. 83–105 (2002).
[16] Lustre: Lustre ADIO collective write driver, Technical report, Lustre (2008).
[17] Ching, A., Choudhary, A., keng Liao, W., Ward, L. and Pundit, N.: Evaluating I/O Characteristics and Methods for Storing Structured Scientific Data, *Proceedings 20th IEEE International Parallel and Distributed Processing Symposium*, IEEE Computer Society, p. 49 (2006).
[18] Luo, X., Mueller, F., Carns, P., Jenkins, J., Latham, R., Ross, R. and Snyder, S.: ScalaIOExtrap: Elastic I/O Tracing and Extrapolation, *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IPDPS, IEEE, pp. 585–594 (2017).
[19] Watkins, N., Jia, Z., Shipman, G., Maltzahn, C., Aiken, A. and McCormick, P.: Automatic and Transparent I/O Optimization with Storage Integrated Application Runtime Support, *Proceedings of the 10th Parallel Data Storage Workshop*, PDSW '15, ACM, pp. 49–54 (2015).
[20] Bauer, M., Treichler, S., Slaughter, E. and Aiken, A.: Structure Slicing: Extending Logical Regions with Fields, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, IEEE Press, pp. 845–856 (2014).
[21] Pedretti, K., Vaughan, C., Barrett, R., Devine, K. and Hemmert, K. S.: Using the Cray Gemini Performance Counters, *2013 Cray User Group Meeting (CUG)* (2013).
[22] Zimmer, C., Gupta, S., Ver ´ onica G. VergaraLarrea : Finally, A Way to Measure Frontend I/O Performance, *2016 Cray User Group Meeting (CUG)* (2016).