Vol.2017-HPC-162 No.12
2017/12/18

# 「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

辻田 祐一[1,a)]　古谷 吉隆[2]　肥田 元[3]　山本 啓二[1]　宇野 篤也[1]　末安 史親[2]

**概要**：計算アプリケーションの大規模化に伴い，そこで扱うデータ量も増加の一途を辿っており，スーパーコンピュータにおけるファイルシステムに対しても，高い性能が求められるようになってきている．スーパーコンピュータ「京」（以下，「京」）のファイルシステムには，Lustre をベースに富士通により機能拡張が行われた並列ファイルシステムである Fujitsu Exabyte File System (FEFS) が用いられ，大規模かつ高速なファイル I/O が可能になっている．しかしながら，プロセス個々にファイルを大量に生成しているジョブも散見され，ファイルアクセスのパターンによっては，ファイルシステムの高負荷やレスポンス低下を招き，当該ジョブ自身のファイル I/O の性能低下だけでなく，他のジョブや「京」の運用へ影響を与える恐れもあるため，運用上改善すべき事象として対策を検討している．我々はそのような高負荷あるいはレスポンス低下を招いているジョブの分析を行い，FEFS の効果的な利用方法に繋げるための検証試験を行った．本稿では，ファイルシステムの I/O 性能改善に取り組む背景や，ジョブ実行情報からの高負荷や低レスポンスを招いているジョブの分析について述べた後に，さらに高負荷や低レスポンスを招く I/O パターンの検証と FEFS の QoS 機能の有効性を確認した性能評価結果について報告する．

**キーワード**：スーパーコンピュータ「京」，FEFS，MDS，OSS，ストライピング

## 1. はじめに

スーパーコンピュータ「京」 [1]（以下，「京」）は，2012 年 9 月の共用開始から 5 年以上が経過し，これまでにハードウェア・ソフトウェアの機能改善等が進んできた半面，近年はファイルシステムの障害や問題が顕在化してきている [2, 3]．その中には，ジョブ中のファイル I/O に起因するファイルシステムの高負荷あるいはレスポンス低下がある．このようなケースでは，ジョブ自身のファイル I/O 性能が低下するだけでなく，他のジョブのファイル I/O の性能低下も招いている恐れがある．また，高負荷な状態が著しかったケースでは，ファイルシステムの停止に繋がるような事象もこれまでに発生しており，運用改善に向けた早急な対応が求められている．ファイルシステムの高負荷やレスポンス低下を招く恐れのあるジョブを事前に見つけ出すことや，そのようなジョブを極力減らす目的において，ファイルシステムの高負荷やレスポンス低下を招くファイル I/O の特徴を調査・分析し，性能低下を招く事例に関するユーザへの情報提供や，「京」の運用における監視機能の強化に資することは，運用改善の取り組みとして有用であると考えている．その観点から，我々は，これまでのジョブ実行履歴やファイルシステムの運用履歴から，性能低下を招いていた I/O パターンの特徴の抽出と，それに基づく検証試験を実施した．

ファイルシステムの Meta Data Server（以下，MDS）における負荷や I/O 要求の処理待ちのキューの滞留状況などを調べたところ，大きく分けて 2 種類の問題に分類できた．一つ目は，大量のファイル生成を同時に行うことによる MDS 高負荷状態で，これは MDS の処理能力を越えた I/O 要求がクライアントから送られて発生する．二つ目は，大量のファイル生成を同時に行う際に，ファイル数とストライプカウントの値の積が過剰に大きくなりすぎる際に起きる MDS のレスポンス低下である．代表的なベンチマークである MDTEST [4] 及び IOR [5] を用いてそのような I/O パターンに関する性能評価を実施した．その結果，「京」の運用情報で得られていたファイルシステムの高負荷あるいはレスポンス低下の状態が再現され，問題となるファイル I/O パターンの確認ができた．現在は，この情報に基づいた高負荷あるいはレスポンス低下を招く恐れのあるジョブの監視機能の強化や，ユーザへの情報提供に向けた取り組みを順次開始しているところである．さらに，FEFS が有する QoS 機能によるファイルシステムの性能改善の可能性も性能評価を通して確認できた．以下，第 2 章では，「京」

[1] 国立研究開発法人理化学研究所　計算科学研究機構
[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

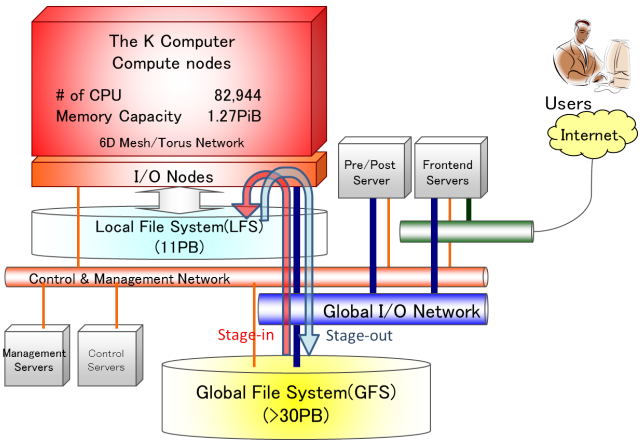



**図 1** 「京」の構成

の運用で確認された，ファイルシステムの高負荷あるいはレスポンス低下を招いたジョブの特徴の分析について述べる．次に，第 3 章において，今回の性能低下を招く要因となったファイルシステム内部の実装メカニズムと性能低下との関連について説明する．さらに，今回対象としている性能低下を招いたファイル I/O パターンに関する検証のために行ったベンチマークによる性能評価結果を第 4 章で報告する．第 5 章において，前章の性能評価結果等も踏まえた上での，今後のファイルシステムの性能改善に向けた取り組みの可能性について述べた後に，最後に，本稿のまとめを第 6 章で行う．

## 2. ファイルシステムの高負荷・レスポンス低下を招くジョブの特徴の分析

「京」は，**図 1** に示すように，ユーザのプログラムやデータを保管する容量重視の Global File System（以下，GFS）と，計算中の高速なファイル I/O を実現する性能重視の Local File System（以下，LFS）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．GFS と LFS の間は I/O 用のノード（以下，I/O ノード）を介して InfiniBand（QDR）および「京」のノード間インターコネクトである Tofu [6] により接続されており，GFS と LFS の間でファイルの転送処理を他のジョブの実行中に平行して行う非同期のステージング機構 [7,8] を採用している．これによって，全体のジョブ実行効率が大きく向上している．

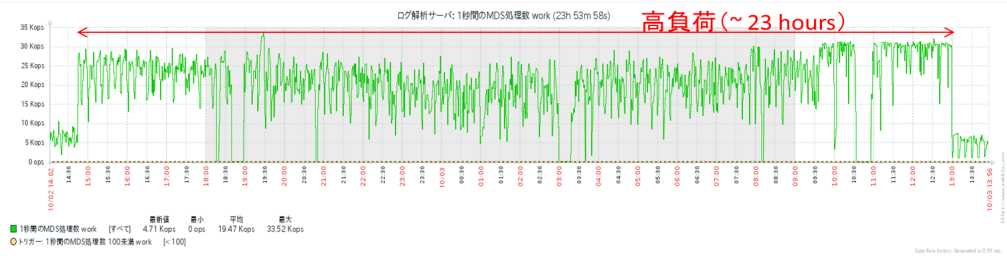
ジョブ実行に関しては，ステージインと呼んでいる GFS から LFS へのファイル転送処理の後，計算ノード群による計算処理が行われ，計算中のファイル I/O には LFS が使用される．「京」における LFS に対するファイル I/O では，全プロセスから共通にアクセス可能な共有ディレクトリと，プロセス個々に独立にアクセスでき，他のプロセスからの干渉を受けないランク番号ディレクトリの 2 種類が利用できる．ランク番号ディレクトリでは，loopback デバイスを用いたファイルシステム上に形成されるため，LFS の MDS へのアクセスが発生せず，特にプロセス数が多く，プロセス個々でファイルアクセスをする場合に有用である．ジョブ実行後，ユーザが明示的に指定したデータ等が，ステージアウト処理により LFS から GFS へファイル転送される．

ステージングも含めたファイル I/O 処理全体の性能低下の問題を考えたときに，特にジョブ実行時の LFS における障害等に起因するファイル I/O の遅延は，ジョブ実行の経過時間の観点からもインパクトが大きい．LFS におけるファイル I/O において，プロセス毎にファイルを生成する場合などファイルシステムの高負荷を招きやすいケースでは、運用側としては，ランク番号ディレクトリの使用を推奨している．しかしながら，割当て可能なノードあたりのディスク容量が 100 GiB までとなっており，ノード内に複数のプロセスがあれば、さらにプロセス数で割った容量が各ランク番号ディレクトリの容量となるために，ランク番号ディレクトリでは個々のディレクトリ内に大規模なファイルを格納することが出来ない．そのために，ジョブ実行形態やプログラムの構造上の理由から，やむを得ず共有ディレクトリを使用しているケースも少なからず存在する．

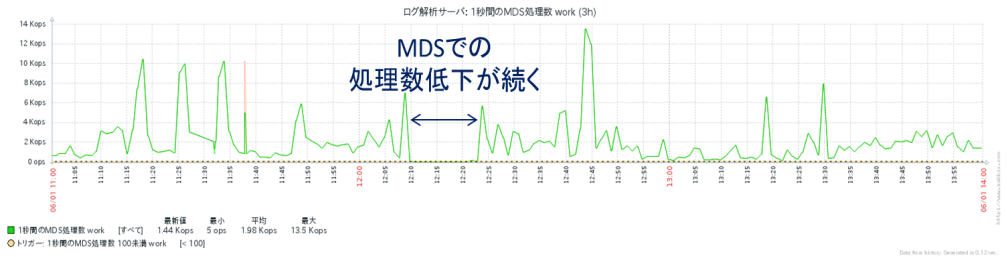
共有ディレクトリでファイル I/O を行う場合には，できる限りファイル数を少なく，かつデータをまとめて一つのファイル内に格納することが望ましい．そのために，例えばランク間でまとめてファイルを生成するような利用形態をお願いしているが，ファイルシステムの特性を十分に理解されずに，プロセス個々に大量のファイルを共有ディレクトリ上に生成するジョブも散見されている．この場合，1 台の MDS に負荷が集中してしまい，MDS の高負荷を招くため，当該ジョブだけでなく，他のジョブのファイル I/O の遅延も招き，場合によってはファイルシステムが不安定になるなど，運用にも大きな支障が出る恐れがある．

非同期ステージングに関しても，上で挙げるようなアクセス遅延の影響は状況によっては無視できないレベルになることが危惧される．また，ランク番号ディレクトリにおいても，ジョブ開始時の loopback デバイスによるファイルシステム作成や，ランク番号ディレクトリのマウント処理の時間が長期化する可能性があり，場合によっては作成あるいはマウントの失敗に繋がる恐れもある．ファイルシステムに不具合が生じたり，I/O 性能が低下したことによるステージング処理やランク番号ディレクトリの作成処理の長期化やタイムアウトによる失敗は，運用全体から見れば，事象としては少ないが，一旦事象が発生すると影響範囲が大きいために，注意する必要がある．

**図 2** に，実運用の中で確認された，ファイルシステムの性能低下が発生した時間帯の MDS の毎秒の処理数（OPS）を示す．図 2(a) では，1,000 ノード（実際にはノード形状 $4 \times 18 \times 14$ となり 1,008 ノードが割り当てられた）を使用して 8,000 プロセスによるプログラム実行を行うジョブの




(a) MDS 高負荷の事例




(b) MDS レスポンス低下の事例

**図 2** LFS 性能低下時の MDS 処理性能の状況

ファイルアクセスによって高負荷になった事例を示している．このジョブはランク番号ディレクトリを使用しておらず，各プロセスから大量に共有ディレクトリ上にファイルアクセスを行った結果，約 23 時間もの間，MDS が高負荷な状態が続いていた．これによって他のジョブのファイル I/O においても大きな遅延が発生していた．

一方，図 2(b) に示すような MDS の処理数が極端に低下し続ける事象もあった．この事例では，1 ノードあたり 1 プロセスの配置で 12,096 ノードを用い，ノード数と同じ数のプロセスが動いており，プロセス毎に共有ディレクトリへのファイルアクセスを行っていたと思われる形跡がジョブスクリプトやステージング処理の履歴，ジョブの実行情報などから確認された．このジョブは 12 時 8 分から 12 時 26 分までの 18 分間実行されていたが，ほぼ重なる時間帯（12 時 10 分から 12 時 23 分）に MDS の処理がほとんど行われておらず，この時間帯にレスポンス低下が起きており，同じ時間帯に動いていた上記のジョブのファイル I/O による影響であったことが分かった．

以上，ファイルシステムの性能低下を招く事象において MDS 性能に着目すると，共有ディレクトリにアクセスするジョブに関して，大きく分けて以下の 2 つに分類される．

- 各ランクから大量のファイルアクセスを行うことによる MDS の高負荷（以下，MDS 高負荷）
- 各ランクから大量のファイルアクセスを行うことによる MDS のレスポンス低下（以下，MDS レスポンス低下）

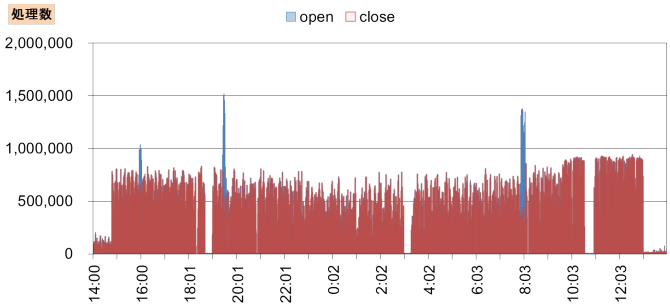
以上の 2 種類のケースについて，ファイルシステムの性能低下を招く要因を検証すると共に，それらを模擬したファイルアクセスパターンに関してベンチマーク等により，I/O 性能への影響を検証した．

## 3. ファイルシステムの性能低下を招いた要因

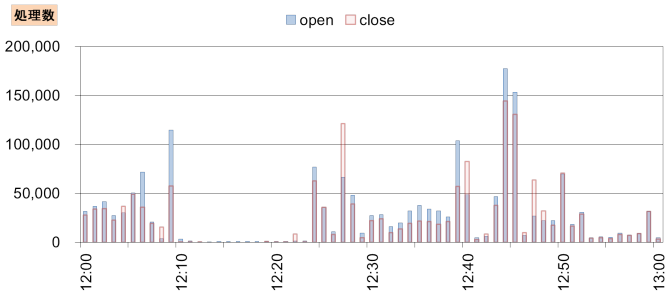
「京」で用いられているファイルシステムである FEFS [9] は，富士通によって Lustre [10] をベースに独自の機能拡張を行ったものである．本章では，第 2 章において述べた性能低下を招く 2 種類のファイルアクセスパターンに関して，性能低下を招いた要因を探るために，MDS 周りを中心にファイルシステムの機能の検証と，当該ジョブが動いていた際のファイルシステムでの I/O 要求の処理状況について確認した．

まず，MDS 高負荷については，共有ディレクトリ上に各プロセスから大量のファイルをアクセスする際に発生することが多い．今回のような，各ランクから大量のファイルアクセスが起きた場合，FEFS を含め，Lustre においては，MDS 高負荷を招いてしまう．また FEFS においては Lustre version 1.8 をベースに実装されているため，1 台の MDS 運用となっているが，これが MDS へのアクセス集中を発生しやすい要因にもなっており，上記のような共有ディレクトリ上へのファイルアクセス集中が高負荷状態に繋がりやすい．

次に，MDS レスポンス低下に関しては，Lustre に起因する問題であることが分かった．Lustre や Lustre をベースに開発された FEFS においては，ファイル生成時などにクライアントから MDS に I/O 要求が届いた後に，ストライプカウント数に対応する数の処理要求が Object Storage Server（以下，OSS）群へ送信され，MDS 側は，OSS 群からのレスポンスを受けて次の処理に進む．共有ディレクトリ上で大量のファイルを生成・アクセスされる際に，ファイル数とストライプカウントの値の積が過剰に大きな値になっている場合，MDS から OSS への処理要求数が過剰

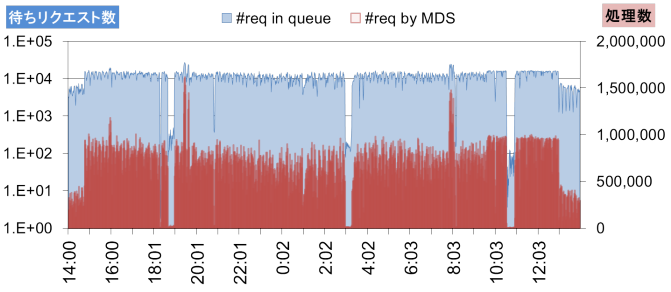



(a) MDS 高負荷の事例（横軸は 24 時間）

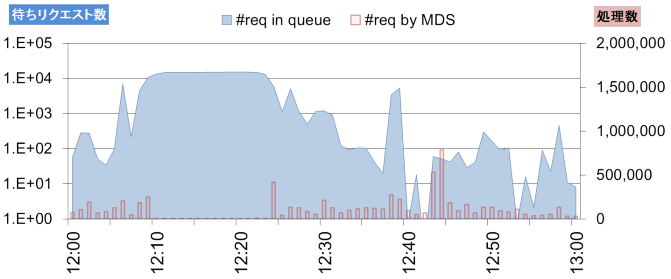



(b) MDS レスポンス低下の事例（横軸は 1 時間）

**図 3** 図 2 で示した 2 つの事例における `open` および `close` の処理数の 1 分毎の推移




(a) MDS 高負荷の事例（横軸は 24 時間）




(b) MDS レスポンス低下の事例（横軸は 1 時間）

**図 4** 図 2 で示した 2 つの事例における処理待ちのリクエスト数と処理数の 1 分毎の推移

になり，各 OSS や各 OST での処理が長期化することで，MDS における OSS 群からのレスポンス待ちが長期化する．その結果，MDS が新たな I/O 要求を受け付けにくい状態になってしまい，レスポンス低下が顕在化したと考えられる．この事例で示したジョブの場合，ストライプカウントはデフォルト設定の 12 を使用していたが，12,000 ノード上に同じ数のプロセスを起動し，プロセスあたり 7 個のファイルにアクセスするために約 84,000 個のデータファイルをステージインさせていた．ジョブの実行時間から，短時間の間に全プロセスから同時期に上記のステージインさせたファイル群へのアクセスを行ったものとみられ，その結果，MDS のレスポンス低下に繋がったと考えられる．

図 2 で示した 2 つの事例に関して，MDS で処理された `open` および `close` の処理数の 1 分毎の推移を確認したものを**図 3** に示す．この図において，推移を見やすくするために，図 3(a) に示す MDS 高負荷の事例と図 3(b) に示す MDS レスポンス低下の事例で縦軸のスケールを変えている．MDS 高負荷の事例では，高負荷になっている 23 時間の間，`open` および `close` 共に，非常に多くの処理が継続して行われていた様子が伺える．一方，MDS レスポンス低下の事例では，レスポンス低下を招いたジョブがファイルアクセスをしていた時間帯（12 時 10 分から 12 時 23 分）に処理数が極端に減少している様子が確認できた．

これらの事象に関して，さらに MDS の処理状況を確認するために，MDS での要求キュー内の処理待ちのリクエスト数と処理された要求数に関して，1 分毎の推移を確認したものを**図 4** に示す．図 4 においては，同じグラフ内で待ちリクエスト数と処理数を同じグラフ上で比較をしやすくするために，左側の縦軸の待ちリクエスト数を対数表示にしている．図 4(a) に示す高負荷のケースでは，待ちリクエスト数がほぼ一定に推移していると共に，処理数も一定量の処理が継続的に行われていることから，MDS が高負荷になりつつも，継続的に I/O 処理が進んでいたと推測される．これは，図 3(a) で示した `open` および `close` の処理数の推移とも一致する．

一方，図 4(b) に示すレスポンス低下のケースでは，レスポンス低下の要因となったジョブがファイルアクセスをしていた時間帯（12 時 10 分から 12 時 23 分）には処理数が極端に減少した状態が続き，その一方で待ちリクエスト数がほぼ一定で推移していた．また，この時間帯の MDS には CPU 負荷はほとんどかかっていなかったことから，OSS 群からのレスポンス待ちが長期化してしまい，その間は新たな I/O 要求を受け付けることがほとんど出来ていなかったものと考えられる．図 3(b) で示した処理数の推移は，このような状況によって起きていたものと考えられる．

「京」の LFS でのファイル I/O においては，ノード形状によって割り当てられる OST の数が一意的に決まる．例えば 192 ノードを確保するためにノード数のみ指定した場合，ジョブスケジューラ側で，その都度配置可能な形状にアレンジされる．3 次元レイアウトで $2 \times 3 \times 32$ でノードを確保した場合，12 個の OST が確保されるが，$8 \times 6 \times 4$ の場合には 96 個の OST が確保される，というように確保される OST の数が形状に依存する．さらに，FEFS では Lustre と同様にストライプカウントに-1 を指定すると，割り当てられた OST 全てでストライプアクセスをする設定

になる．これをユーザが定常的に使用しているケースもこれまでに散見されており，その中には，プロセス毎に大量のファイルアクセスを行った結果，アレンジされたノード形状により，ストライプカウント数が大きくなり，レスポンス低下を招いていた事象も確認されている．

以上のことから，MDS の高負荷状態に関しては，これまでに整備した監視機能で容易に検出できる体制を整えているが，一方で後者のレスポンス低下は，MDS の負荷だけからでは一見安定して稼働しているように見えてしまうため，ファイル I/O 性能低下の状況を見逃してしまう恐れがある．その観点で，本稿では特に後者の事象の特徴の抽出を兼ねた性能評価を中心的に行った．

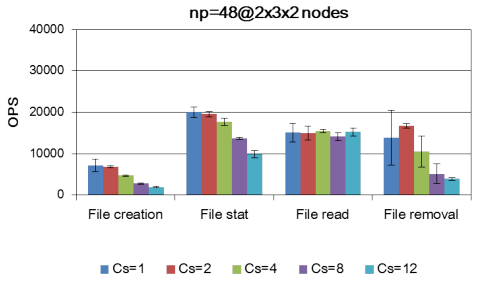
## 4. 性能評価

これまでの運用情報から確認された MDS 高負荷および MDS レスポンス低下に伴うファイルシステムの性能低下の影響を確認するために，代表的なベンチマークである MDTEST および IOR を用いて，「京」のファイルシステムの性能評価を行った．

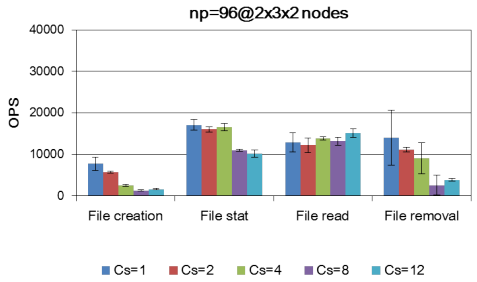
### 4.1 MDTEST による評価

LFS の MDS に対するストライプカウントの影響を確認するために，他のユーザのジョブに影響が出ない範囲で 12 ノード（$2 \times 3 \times 2$）および 48 ノード（$2 \times 3 \times 8$）の計算ノードを用い，1 ノードに 4 プロセスおよび 8 プロセスを配置するレイアウトで MDTEST によるローカルファイルシステムに対する性能評価を実施した．ベンチマークのパラメタとしては，ランク毎に別々のディレクトリでアクセスする環境下で，各プロセスあたり 100 個のファイルをアクセスする設定で，3 回の繰り返しから得られた計測結果を**図 5** に示す．図中の $C_S$ はストライプカウントを示しており，割り当てられた 12 個の OST に対して 1 から 12 までの 5 ケースのストライプカウント設定を評価した．計測したファイル関係の 4 項目のうち，File read の処理以外でストライプカウント増加に伴うレスポンス低下が確認できた．また，1 ノードあたりに起動するプロセス数を増やした場合には，性能が若干低下する傾向が確認された．これは `max_rpcs_in_flight` で定められる 1 ノードあたりの RPC 要求の同時発行数が，「京」の計算ノードにおいては 1 に設定されているために，1 ノードあたりのプロセス数を増加させても処理が律速して性能低下に繋がった可能性が考えられる．

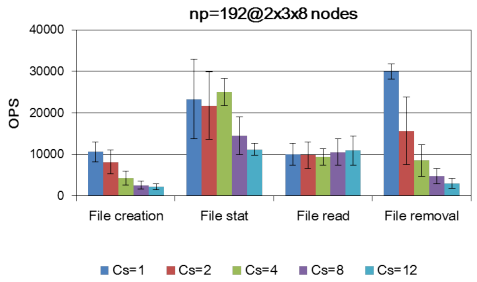
「京」においては，設置時に性能評価を通して `max_rpcs_in_flight` を 1 に決定した経緯があったが，その際の性能評価はノードあたり 1 プロセスの配置であった．よって今回の評価のようにノードあたりに複数のプロセスを配置した場合の最適値が 1 かどうかは現状では不明である．しかしながら，`max_rpcs_in_flight` の値を増やすことは，MDS 側に対して，より多くの要求が短時間に送られる可能性があり，MDS 高負荷や MDS レスポンス低下を回避する意味では，少な目に設定しておいた方が安全とは言える．



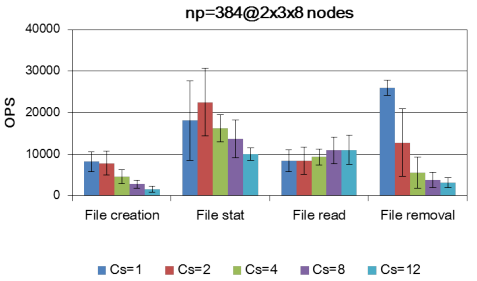

(a) 48 プロセス，12 ノード ($2 \times 3 \times 2$)




(b) 96 プロセス，12 ノード ($2 \times 3 \times 2$)




(c) 192 プロセス，48 ノード ($2 \times 3 \times 8$)

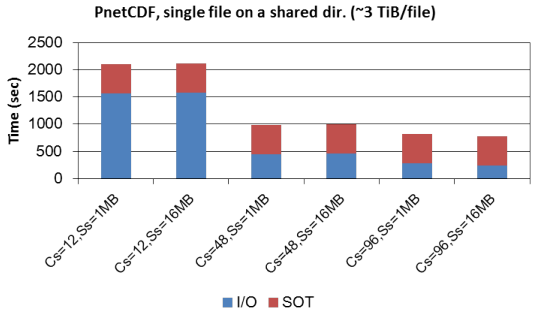



(d) 384 プロセス，48 ノード ($2 \times 3 \times 8$)

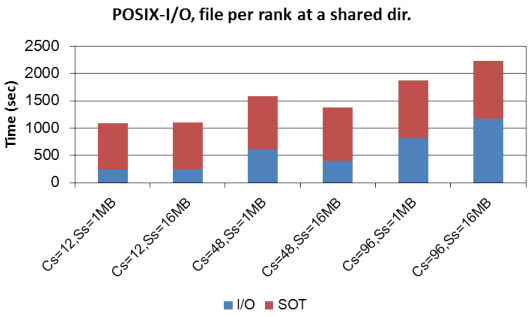
**図 5** ストライプカウント（図中の $C_S$）を変化させた時の MDTEST による計測結果

### 4.2 IOR による評価

1,536 ノード（形状：$8 \times 6 \times 32$）の計算ノードを用い，各ノードに 8 プロセスずつ配置して，計 12,288 プロセスによって IOR による書き込み性能の評価を行った．IOR でのデータ I/O 長とブロック長は共に 256 MiB とし，セ

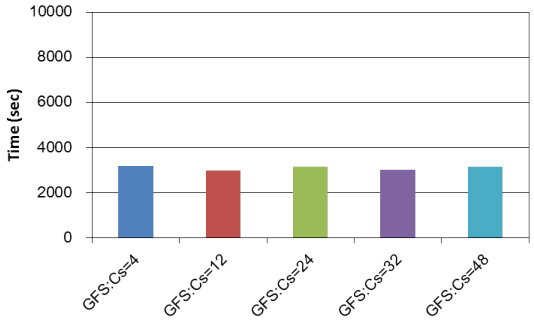



(a) PnetCDF による集団型 I/O

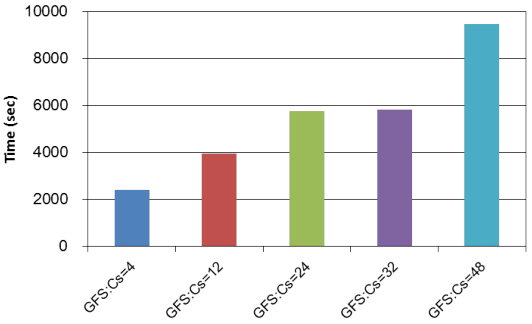



(b) POSIX-I/O によるプロセス毎のファイルアクセス

**図 6** IOR による集団型 I/O 並びにプロセス毎でのファイル書き込み処理の性能．I/O および SOT は，それぞれ IOR での書き込み処理およびステージアウトに要した時間を表す．




(a) 集団型 I/O での共有ファイル生成（3 TiB/ファイル，計 3 ファイル）




(b) プロセス毎のファイル生成（256 MiB/ファイル，計 36,864 ファイル）

**図 7** GFS 側のストライプカウントを変更した時の集団型 I/O でファイル生成したケースおよびランク毎に生成されたケースでのステージアウト時間

グメント数を 1 とした．これにより，プロセス全体で 1 個のファイルあたり 3 TiB のデータ長となる書き込み処理を 5 回繰り返し，I/O 時間の平均値を算出した．IOR での測定においては，PnetCDF [11] を用いた集団型 I/O と POSIX-I/O によるプロセス毎に別々のファイルへ書き込みを行う 2 通りのケースを用い，アクセスする共有ディレクトリのストライプサイズについてはデフォルトの 1 MiB と 16 MiB の 2 ケースので評価した．なお，PnetCDF に関しては，version 1.6.1 を用い，「京」の MPI-IO ライブラリ上に構築したものを用いた．

ステージアウト先の GFS のストライピング設定は，「京」の運用で現在採用しているデータ長に基づいたストライプカウント設定を用いた．そのため，ストライプカウントに関しては，PnetCDF を用いた集団型 I/O ではファイルサイズが大きいため，設定可能な最大値の 32 を，POSIX-I/O によるプロセス毎に独立したファイル生成をしたケースでは，約 256 MiB のデータ長に対応する 17 を用いた．一方，ストライプサイズはいずれもデフォルトの 1 MiB を用いた．

3 回の計測から得られた書き込み時間ならびに生成したファイル群のステージアウトに要した時間を**図 6** に示す．この結果から，集団型 I/O により共通のファイルへ書き込み処理を行う場合，データサイズが大きければ，ストライプカウントを大きめに設定することで I/O 時間の短縮が可能であることが分かる．一方，プロセス毎に別々のファイルへ書き込む場合には，プロセス数の増加に伴いファイル数が多くなることから，ストライプカウントを小さめに設定する方が処理時間の短縮に繋がる．また，いずれのケースにおいてもステージアウト時間は LFS 側のストライプカウント設定にはほとんど依存せずに一定の時間を要していた．これは，LFS に比べて，GFS の方が性能が低いために LFS 側のストライプカウントの影響が表れにくいことによるものと考えられる．

そこで，逆に GFS 側のストライプカウント設定を変えてステージアウト時間を計測した．ここでは，LFS 側のストライプカウント及びストライプサイズはそれぞれデフォルトの 12 並びに 1 MiB を用いた．計測結果を**図 7** に示す．集団型 I/O のケースではストライプカウントに依存せずにほぼ一定時間であるのに対し，ランク毎のファイル生成では，ファイル数が多くなり，ストライプカウントの増加に伴い，ステージアウト時間が長期化してくる様子が確認できた．過剰なストライプカウント設定がファイルアクセス時のレスポンス低下を招くことを考慮すると，ファイル数やファイルサイズに応じて，I/O 性能とのバランスを考えた上で，できるだけ小さめのストライプカウントを設定することが望ましいとも言える．

### 4.3 MDS における QoS 機能の検証

FEFS が有する QoS 機能を MDS に対して適用した際の

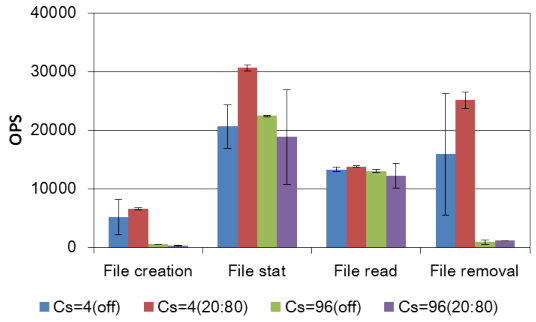



**図 8** QoS 機能の有無による MDTEST による MDS 性能

性能への影響を調査するために，計算ノードを 192 ノード（3 次元レイアウトで $8 \times 12 \times 2$）ずつ 2 グループ（外乱側および計測側）に分けて，それぞれに MDTEST のベンチマークを実行させて共有ディレクトリに対するアクセスにおける MDS の評価を行った．QoS 機能については，既に GFS における大規模ステージングによるレスポンス低下の改善のために，GFS の OSS 群に適用して安定した運用を継続している実績があり [12]，LFS の MDS に対しても同様の効果が期待できる．なお，「京」の LFS の MDS においては，サービススレッド数の上限を一部のサービススレッドを除き，24 に設定して運用している．この値は「京」の設置時の性能評価により最適な値として選択した経緯があったが，第 4.1 章で説明した `max_rpcs_in_flight` を 1 に設定した経緯と同様に，1 ノードに 1 プロセスの配置での評価であった．よって，ノードあたりに複数プロセスを配置した場合に，この値が最適であるかは不明である．しかしながら，1 台の MDS 上に過剰にスレッドを立ち上げることによるスレッド間の処理の混雑による処理の遅延や，計算ノード側で `max_rpcs_in_flight` を 1 に設定している状況を踏まえると，最適とは言えないかもしれないが、現状においては妥当な設定であると考えている．

MDS に対して QoS 機能の設定を行うことから，通常のジョブ実行期間ではなく，「京」のメンテナンス時間の中の空いている時間帯に実施した．ここで，一方は外乱側として 1 ノードに 1 プロセスずつ計 32 プロセスを配置し，もう一方のグループには計測側として 1 ノードに 4 プロセスずつ計 768 プロセスを配置した．外乱側で継続的にベンチマークを実行させている間に，計測側で各ケースで 3 回の繰り返しで，プロセス毎に別々のディレクトリを作成し，その中で 100 個のファイルの生成を行った．

QoS 機能に関しては，設定していない場合と，外乱側と計測側をそれぞれ 20%並びに 80%の割合で MDS のサービススレッドを割り当てる場合において，各々でストライプカウントを 4 および 96 に設定して計測を実施した．計測結果を**図 8** に示す．ストライプカウントが 4 の時は QoS 機能を使わない場合（図中の $C_S$=4(off)）に比べ，QoS を有効にした場合（図中の $C_S$=4(20:80)）の方が”File stat”あるいは”File removal”の操作で性能が向上している傾向が見られる．一方，ストライプカウントを 96 にすると，QoS なし（図中の $C_S$=96(off)）と QoS あり（図中の $C_S$=96(20:80)）とで，大きな性能差は見られなかった．これは計測側に対応している MDS のスレッド群がレスポンス低下している状況では，逆に計測側が外乱側へ影響を及ぼす側になっており，計測側では QoS による性能向上が実現できていなかったことによると考えられる．そこで，ストライプカウントが 96 の時の MDS におけるリクエストの処理数について調べたところ**図 9** に示すような状況であった．図中の矢印で示したストライプカウントを 96 で 3 回繰り返して評価していた時間帯において，外乱側のみ動作し処理数が跳ね上がった時間帯を除いては，QoS が無い場合（図中の $C_S$=96(off)）に比べて，QoS を有効にした場合（図中の $C_S$=96(20:80)）の方が処理数が多くなっている傾向が見られる．この結果から外乱側での MDTEST 評価では QoS 機能による性能向上が実現できていたものと考えられる．つまり，MDS のレスポンス低下を招くジョブがある状態でも，QoS 機能により，他のジョブ（今回は外乱側）での性能低下を低減できる可能性があると言える．しかしながら，「京」のメンテナンス時間の中の限られた時間帯に本試験を実施する必要があったことから，今回のデータだけでは不十分な点もあり，今後も継続して検証を進める予定である．

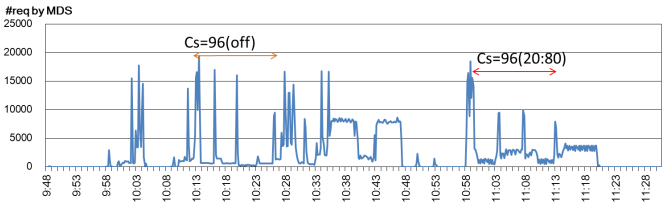



**図 9** MDTEST 実行時の MDS におけるリクエストの処理数の推移

## 5. ファイルシステムの I/O 性能改善に向けた今後の取り組み

既に述べたように，ファイルシステムにおける I/O 性能の低下は，ユーザのジョブの実行時間超過以外にも，ステージング時間の長期化なども招く恐れがあるなど，「京」の運用に大きな影響がある．そのような事態をできるだけ回避あるいは早急に検知し，安定した運用を継続することが重要となる．その観点で，本稿で述べたような高負荷あるいはレスポンス低下を招くようなジョブの特徴を抽出し，ユーザへの周知徹底に努めるだけでなく，運用における適切な対応を進める必要性がある．適切になるべく小さいストライプカウント設定を行う有効性や，FEFS が有する QoS 機能を併用することによる MDS 性能向上の可能性も確認できたが，これについても有効性の検証をさらに進める必要がある．

運用における試みとしては，既にリアルタイムなジョブ単位での MDS への処理数の監視を行っている．また，MDS での I/O 要求の滞留状況や処理数などの定期的な集計機能も既に稼働させている状況である．現在は個々の機能の試験運用中で，手動による監視機能の検証を進めているところであるが，ファイルシステムの高負荷を発生させているジョブの特定においては実用段階に来ており，今後は個々の監視機能を統合させた自動検出機能の強化や，さらにはファイルシステムの運用継続が難しくなるような高負荷状態を招いているジョブの自動停止機能の実装などの運用改善に努めてゆく予定である．

一方，レスポンス低下に関しては MDS の監視だけでは不十分と考えており，例えば並行して OSS 側の監視を行うことが考えられるが，その有効性については今後検証を進める予定である．なお，「京」の LFS に関しては 2,500 台以上もの OSS があり，仮に OSS の監視も有効と判断できたとしても，全 OSS を手動により逐一監視し続けることは現実的に困難であるため，効率的・現実的な手法を検討する必要があると考えている．

## 6. 本稿のまとめ

「京」のファイルシステムの高負荷あるいはレスポンス低下を招くようなファイルアクセスを行うジョブの実態をジョブ実行情報やファイルシステムの統計情報から分析を行うと共に，特徴的なファイルアクセスパターンに対して MDTEST および IOR を用いたベンチマーク評価による実証も行った．共有ディレクトリへのアクセス性能を低下させるジョブの特徴として，各プロセスから大量のファイルアクセスを行うものと，大量のファイルアクセスにおいて，ファイル数とストライプカウントの積が過剰に大きくなっているものの，計 2 パターンがあることを確認した．前者は MDS 自体が高負荷に，後者では MDS 自体はレスポンス低下を招いていたが，後者のケースでは，MDS における OSS 群からのレスポンス待ちが長期化することで，MDS の処理が滞っていたことが分かった．ベンチマークによる検証試験でも同様の状態が確認でき，ファイルシステムの性能低下を招く恐れのあるジョブの特徴をある程度特定することができた．現在は，ここで得られた知見から，ファイルシステムの性能低下を招く可能性のあるジョブの特定や，そのようなジョブを実行しているユーザに対し，ランク番号ディレクトリの使用や，適切なストライプカウント設定をお願いするなど，性能改善に努めている．

ジョブ実行中のファイル I/O の遅延はジョブ実行時間の長期化など，インパクトが大きく，ここでのファイル I/O 性能の改善が最も重要であるが，それ以外にもステージング処理時間の長期化の可能性もある．ステージングで処理されるファイル数やデータ量が多くなれば，その影響はより顕著になり，非同期ステージングといえども，ある程度の影響が懸念される．またファイルシステムの高負荷やレスポンス低下はランク番号ディレクトリの生成時間の長期化や失敗にも繋がるため，今回報告したファイルシステムの性能改善の取り組みをさらに強化してゆく必要があると考えている．今後の課題としては，MDS の高負荷やレスポンス低下を検知する監視機構の強化が挙げられる．現在でも MDS の負荷の監視や個々のジョブの MDS へのアクセス量の監視は行っているが，将来的に，高負荷を招いているジョブの自動停止機能などの環境整備を進めてゆく予定である．また，LFS の高負荷あるいはレスポンス低下時においても，計算中のファイル I/O やステージング等の処理に与える影響を小さくする意味で，今回実施した検証試験結果を受けて，QoS 機能の実運用への適用も検討する予定である．

**謝辞** 本論文の結果は，理化学研究所のスーパーコンピュータ「京」を利用して得られたものであり，利用に際して富士通株式会社からは技術的なご支援を頂きました．ここに感謝の意を表します．

### 参考文献

[1] : 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).
[2] Shoji, F.: Lessons learned from development and operation of the K computer, *Parallel Computing*, Vol. 64, pp. 12–19 (online), DOI: https://doi.org/10.1016/j.parco.2017.03.001 (2017).
[3] 宇野篤也，肥田 元，関澤龍一，辻田祐一，山本啓二：「京」のファイルシステム障害の分析，情報処理学会研究報告，Vol. 2017-HPC-160, No. 24, pp. 1–6 (2017).
[4] MDTEST: `https://sourceforge.net/projects/mdtest/`.
[5] IOR: `https://sourceforge.net/projects/ior-sio/`.
[6] Ajima, Y., Inoue, T., Hiramoto, S., Takagi, Y. and Shimizu, T.: The Tofu Interconnect, *IEEE Micro*, Vol. 32, No. 1, pp. 21–31 (2012).
[7] 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).
[8] Hirai, K., Iguchi, Y., Uno, A. and Kurokawa, M.: Operations Management Software for the K computer, *Fujitsu Sci. Tech. J.*, Vol. 48, No. 3, pp. 310–316 (2012).
[9] 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).
[10] Lustre: `http://lustre.org/`.
[11] Parallel netCDF: `http://cucis.ece.northwestern.edu/projects/PnetCDF/`.
[12] Tsujita, Y., Yoshizaki, T., Yamamoto, K., Sueyasu, F., Miyazaki, R. and Uno, A.: Alleviating I/O Interference Through Workload-Aware Striping and Load-Balancing on Parallel File Systems, *High Performance Computing - 32nd International Conference, ISC High Performance 2017, Frankfurt, Germany, June 18-22, 2017, Proceedings*, Lecture Notes in Computer Science, Vol. 10266, Springer, pp. 315–333 (2017).