Vol.2019-HPC-168 No.21
2019/3/7

計算ノードの使用効率向上を目指した「京」のファイルシステムの運用改善

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

概要：スーパーコンピュータ「京」（以下，「京」）の計算ノードの使用効率向上を目指して，我々はファイルシステムの運用改善に取り組んでいる．「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．多数のクライアントが同一ディレクトリへのファイルアクセスを行うと，ファイルシステムの処理効率が著しく低下する．処理効率が低下すると，経過時間を超過したジョブの削除措置が，ファイルシステムの問題で適切に処理されない．その間計算ノード群は利用できないため，現状では制限時間内にジョブの削除が完了しない場合，その計算ノードを停止させて処理を完了させている．ジョブ削除やノード復旧の待ち合わせで計算ノードの使用効率が低下するため，更なる改善が求められる．本稿では，このような使用効率を低下させる問題に対して，Lustre が有するクライアント追放機能（evict）に着目し，I/O リクエストを中断させることでノード時間積の損失を低減させる方法を提案する．「京」の 6 万台の計算ノードを用いて評価を行い，提案手法の有効性を確認した．

キーワード：Lustre，evict，MDS，I/O リクエスト中断，ノード時間積

1. はじめに

近年，HPC システムは多様化かつ大規模化し，それに伴い扱うデータ量は増加の一途を辿っている．それらのデータを管理するファイルシステムには，高性能かつ大規模であると共に，高い安定性が求められるため，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．Blue Waters や「京」などの大規模な HPC システムにおいて，システム障害の多くはファイルシステムに起因して発生している [3,4]．このように，システムの規模や構成に起因して顕在化する問題が存在する．「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, これまでそのようなファイルシステムの問題に対して継続的に運用改善 [5,6] を行っているが，現在でも計算ノードの使用効率に影響を与える障害事象が散見されている．

計算ノードの使用効率低下を招く障害要因として，多数のクライアントから行われる同一ディレクトリへのファイルアクセスが挙げられる．同一ファイルやディレクトリに対する一斉アクセスが行われると，ファイルシステムへの I/O リクエストが滞留し，ジョブが予定した経過時間内に終了しない．経過時間を超過したジョブをスケジューラが削除しようと試みるが，ファイルシステムの問題でジョブの削除処理が長期化してしまう．現在の運用では，ジョブが削除できず一定の時間を経過した計算ノードについては，強制停止後に再起動させることで対象ジョブを削除し，後続のジョブが計算ノードを利用できるようにしている．ジョブが割当たった計算ノード群はジョブ削除まで利用できないことに加え，削除できなかった計算ノードに至っては，さらに停止・再起動をする間も利用できない．そこで，Lustre が有するクライアント追放機能（evict）に着目し，I/O リクエストを中断させることを考える．I/O リクエストが完了せず削除できないジョブに対して，evict 機能を用いて処理を中断させノード時間積の損失を低減させる手法を提案する．

第 2 章で「京」のファイルシステムの概要を紹介する．第 3 章で問題となる I/O リクエストと運用影響，第 4 章でシステム的に問題を回避するための改善案を示し，第 5 章で改善案の評価を行う．関連研究について第 6 章で述べ，最後に第 7 章でまとめを行う．

2. 「京」のファイルシステム

「京」は，Lustre ver. 1.8 をベースとして富士通が独自に

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

© 2019 Information Processing Society of Japan

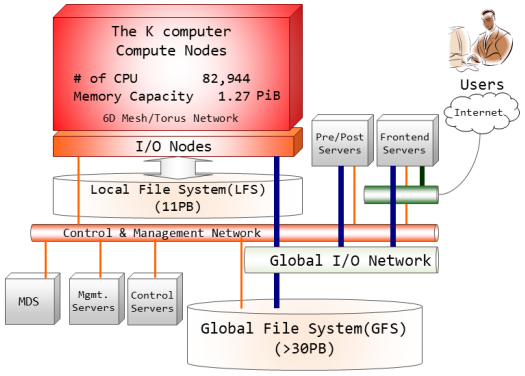

図 1 「京」のシステム構成

機能拡張を行った FEFS (Fujitsu Exabyte File System) [7] を用いている．ファイルシステムは，大別するとローカル・ファイルシステム（以下，LFS）とグローバル・ファイルシステム（以下，GFS）の 2 つから構成される．「京」のシステム構成を図 1 に示す．

LFS は，アプリケーションからの I/O 性能を確保することを目的としたファイルシステムで，計算時の入力データや結果の出力など，アプリケーションの計算の為だけに用いられる．LFS は，全計算ノード 82,944 台からマウントされており，アプリケーションを実行するノード間でデータの共有が可能である．一方，GFS は利用者のアプリケーションやデータファイルの保管領域として利用される．「京」では，必要なアプリケーションやデータファイルを GFS から LFS に転送して，アプリケーションの実行を行う [8]．ファイルの転送は，ジョブの実行とは非同期に行われるため，効率的に計算資源が利用される．

3. 問題背景

本章では，「京」で発生した長期化する I/O リクエストと，発生時の運用影響を報告する．運用中に問題となった同一ディレクトリへのファイルアクセスのパターンは 2 つある．一つは同一のディレクトリ配下への一斉ファイル作成で，もう一つは切り詰めを行う truncate 処理や書き込みを同一のファイルに行う操作である．このうち，今回は運用影響の大きかったファイル作成処理を対象する．

3.1 長期化する I/O リクエスト

同一ディレクトリ配下へのファイル作成は，クライアント間のキャッシュの一貫性保証のため，クライアント毎にシリアライズされた処理となる．ファイルアクセスのメタ情報を管理するメタデータサーバ (MDS) は，リクエスト処理のためのスレッドを複数持つが，このケースでは同時に動作が行えないため，処理が長期化する．処理が長期化した場合，クライアントが I/O リクエストを送信してから一定時間が経過すると，クライアントは自身の送信したリクエストのタイムアウトを検出し，サーバに再接続を行い，

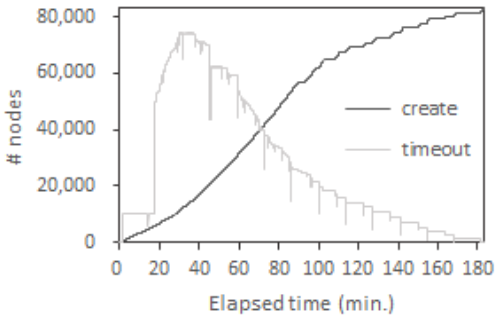

図 2 同一ディレクトリへのファイル作成状況

リクエストを再送する．一方サーバ側は，処理の長期化でタイムアウトしたリクエストを破棄し，クライアントからの再送を待つ．

図 2 に，82,944 台の計算ノードから同一ディレクトリへファイル作成した際の，作成が完了した計算ノード数（図中の create）と，リクエストがタイムアウトし再送を行った計算ノード数（図中の timeout）を示す．この事象においては，1 ノードに 1 プロセスずつ起動しており，合計で 82,944 個のファイルを作成しようとしていた．このファイルは，アプリケーションの標準出力およびエラー出力を各プロセス毎に受け取る目的で作成され，全て同一ディレクトリ上に作成された．リクエストは到着順に処理されファイルが作成されたが，到着が遅れたリクエストはキュー内で待たされ，タイムアウトに至った．

図 2 の経過時間が 51 分時点でジョブの削除コマンドを投入したが，ジョブが解放され，後続ジョブが走行可能となったのは経過時間が 185 分時点であった．ジョブの削除コマンドでジョブプロセスにシグナルが送信されたが，ファイル作成処理が停止できずリクエストの再送が行われ続けていた．これは FEFS のベースとなる Lustre の仕様で，Lustre の特定のコンポーネント（ptlrpc）まで処理が進むと，SIGKILL を含む全てのシグナルをブロックするためである．Lustre におけるクライアント・サーバ間の処理フローの概要を図 3 に示す．一度依頼されたリクエストは基本的に中断されず，タイムアウトしたリクエストは ptlrpc 層で再送されるため，上述のような事象が発生した．

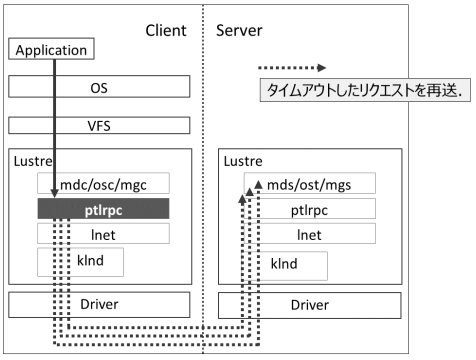

図 3 Lustre における I/O 処理フロー概要

© 2019 Information Processing Society of Japan

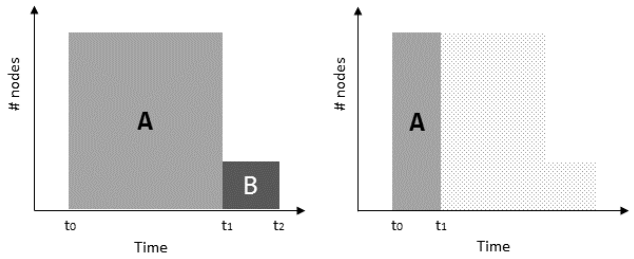

図 4　ジョブの削除処理で失うノード時間積
（左）I/O を待ち合わせる場合（右）I/O を中断する場合

3.2　現行運用における計算資源の損失

ジョブのプロセスの削除が完了しない計算ノードでは，後続ジョブは実行できないため，計算ノードの利用効率が低下する．そこで，ジョブの削除依頼を受け，計算ノードがジョブプロセスにシグナルを送信してから，一定時間経過しても削除できない場合，計算ノードを異常と判断して強制停止する運用を行ってきた．

ジョブが削除され始める時刻を t_0，削除完了し後続ジョブが実行可能になる時刻を t_1，ジョブが削除できないため停止させたノードの復旧時刻を t_2 とする．ジョブが早期に削除できない場合に失われるノード時間積を図 4 に示す．ここで，A および B は，それぞれジョブプロセスの削除およびジョブプロセスが削除できない場合に対象の計算ノードを停止・再起動させることにより失われるノード時間積を表している．現行の運用方式は，この図の左側に示すように一定時間（この図では $t_1 - t_0$）だけジョブプロセス削除処理を待ち合わせ，この間に割り当てられたノード分だけのノード時間積の損失 (A) が発生する．さらに，一定時間内に削除できなかった計算ノード群の停止および再起動が $t_2 - t_1$ の間行われ，B で示されたノード時間積の損失が発生する．特に，ノード（クライアント）数に比例して，I/O リクエストの処理の時間が増加するため，大規模なジョブ実行においての影響が大きい．

図 2 のジョブが走行した際は複数の計算ノードが停止したが，ジョブの I/O リクエストを中断させることができれば，図 4 の右側で示す通り，計算ノードの資源の損失を改善することができる．さらに，計算ノードの停止処理は，復旧までに時間を要することに加えて，計算ノードの再割当てによる GFS から LFS へのファイル再転送やスケジューラへの外乱など，運用への影響が大きいことも問題となる．そのため，システム側で計算ノードを停止させないためにも，I/O リクエストの中断が必要である．

4.　ノード利用効率向上を実現するジョブ削除手法

前章では，クライアントの I/O リクエストが中断できないことが原因で，ジョブプロセスが削除できず，計算資源が浪費されている問題を示した．本章では，その問題の解決策として，リクエストを中断させる方法と，「京」における実装案を述べる．

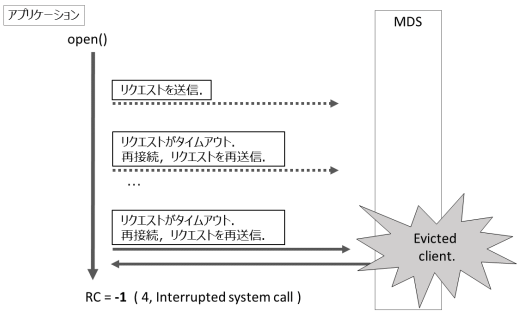

図 5　evict によるファイル作成の中断

4.1　Lustre Client Evictions

evict は，Lustre のサーバが異常と判断したクライアントを切り離す処理のことで，ファイルシステムを利用可能な状態に保つために行われる．例えば，クライアント間で共有するファイルへの排他制御を行うために，サーバはロックの貸し付け・回収を行うが，サーバからの要求に一定時間で応えられない場合，サーバはクライアントを退去させ，ロックの回収を行う．クライアントが退去させられたことに気づくのは，クライアントの次の I/O リクエストがサーバで処理される時である．

evict により長期化したファイル作成リクエストを中断した場合の処理イメージを図 5 に示す．クライアントに evict を通知し，クライアントは未完了の I/O リクエストを全てを放棄する．この仕組みを利用し，中断したいリクエストを送信しているクライアントを evict することで，I/O リクエストを中断する．

4.2　evict によるジョブ削除機能の実装案

機能の概要を図 6 に示す．管理サーバは，ジョブスケジューラのログを監視し，ジョブ削除処理の遅延を検知する．その後，計算ノードに対して I/O リクエストの中断を依頼する．依頼を受けた計算ノードは，ジョブプロセスの残存を確認し，Lustre における自身の NID(LNet Network Identifier) を MDS に通知し，evict を依頼する．処理のタイミングによっては，計算ノードの I/O リクエストが完

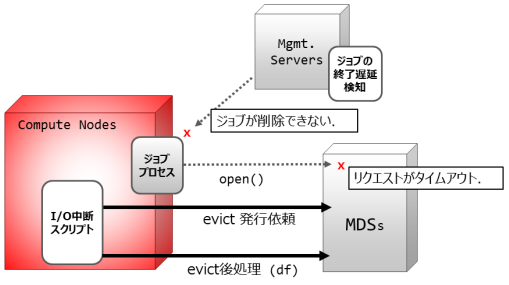

図 6　ジョブプロセスの I/O リクエストの中断機能

© 2019 Information Processing Society of Japan

了した後に，MDS が計算ノードを evict し，次回の無関係な I/O リクエストをエラー返却させてしまう可能性がある．これを防ぐため，計算ノードは evict 依頼後ファイルシステムに一度アクセスを行い，evict された状態を解消する．ファイルシステムへのアクセスは df コマンドを用いる．MDS は，以下に示すコマンド[*1]により，対象クライアントを evict する．

```
# lctl set_param mds.*MDT*.evict_client=nid-mute:${NID}
# lctl set_param fefs_mgs.MGS.evict_client=\
nid-mute:${NID}
```

5. 提案手法の評価

前章で示した evict による I/O リクエストの中断処理について，問題なく動作するか評価試験を行った．

5.1 評価試験

評価にあたり，長期化する I/O リクエストを送信するジョブ (a)，evict 後ファイルシステムが正常に利用できるか確認するジョブ (b) を順に実行した．ジョブ (a) の実行中に，evict 処理およびジョブ (a) の削除依頼を行った．ジョブの概要をそれぞれ次に示す．

ジョブ (a)
60,000 ノード使用．1 ノード 1 プロセスで，LFS 上に配置した MPI プログラムを実行．MPI プログラム実行前に各プロセスの標準出力先となるファイルを同一ディレクトリ上に作成．作成ファイルのストライプカウントは 12.

ジョブ (b)
ジョブ (a) が動作した 60,000 ノードを使用．1 ノード 1 プロセスで，LFS 上に配置した MPI プログラムを実行．

5.2 評価結果

ジョブ (a) がファイル作成を完了する前に削除されることを確認した．またジョブ (b) が問題なく走行したことから，evict 処理後，後続ジョブは正常にファイルシステムにアクセスできることを確認した．**表 1** にジョブ (a) でファイル作成が完了したノード数と提案実装によりファイル作成が中断されたノード数を示すと共に，**図 7** に前者のノード数（図中の create）と，ファイル作成が中断されたノード数を加えたノード数（図中の create+fail）の遷移を示す．図 7 の経過時間が 10 分時点でジョブの削除コマンドを投入し，経過時間 37 分時点でファイル作成と中断したノード数が 60,000 台に到達し，ジョブの削除処理が完了した．

[*1] ${NID} には，対象クライアントの NID を指定．Lustre の場合は，fefs_mgs. の代わりに，mgs. を指定．

表 1 evict によるファイル作成中断の結果

使用計算ノード数	60,000
ファイル作成完了ノード数	52,488
ファイル作成中断ノード数	7,512

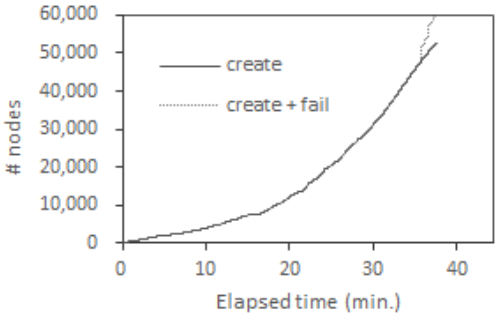

図 7 ファイル作成を完了または中断した計算ノード数

6. 関連研究

HPC システムにおいて，大規模な Lustre ファイルシステムを効率良くかつ安定に運用することは，ジョブの実行効率を高める上で大変重要であり，性能向上や安定稼働を狙った最適化などが研究されている．[9–11].

HPC システムは並列ファイルシステムを含め，様々なコンポーネントにより構成されていることから，計算ノードの故障やファイルシステムの不具合などによる運用効率の低下が避けられない．運用効率向上を目指した運用改善の試みの一つとして，様々なログ出力から機械学習により計算ノード故障を事前に予測する手法が提案されている [12,13]．ただし，これらはログ解析から障害箇所の予測までの処理フローについての試みであり，個々の障害を回避する方法については言及していない．

一方，我々は，これまでの「京」の運用で散見されてきた運用に大きな影響を与えるシステム側が起因となる障害について，ファイルシステムを含めてさらなる運用効率向上を目指して改善に努めてきている．本稿においては，ファイルシステムアクセスが長期化し，経過時間を超過してもジョブが終了できないケースにおいて，利用効率低下を改善する evict によるジョブ停止手法を提案している．これまでのタイムアウト方式と比較し，ノード停止・再起動による利用率低下を防ぐ．事前にファイルシステムを含むシステム側ログから，そのようなジョブを検知し，その情報を元に対象ジョブの強制停止を行う点において，非常に簡易的でかつ容易にノード停止・再起動による損失を防ぐことが出来る点が優位な特徴である．

7. おわりに

大規模な HPC システムにおけるファイルシステムの課題について，「京」での事例とその改善事例について述べた．ジョブの I/O リクエストが長期化する場合，ジョブの削除

© 2019 Information Processing Society of Japan

において，送信されたリクエストの完了を待ち合わせると計算ノードの利用効率が低下するため，Lustre の evict 機能を用いて I/O リクエストを中断する方法を示した．「京」の 6 万台の計算ノードを用いて評価を行い，I/O リクエストを中断してジョブを削除する有効性を確認した．

evict により，計算ノード上で動作しているシステムプロセスの I/O リクエストがエラー返却される可能性がある．「京」では，プロセスごとに独立してアクセス可能な領域（ランク番号ディレクトリ）を loopback デバイスにより提供しており，ジョブの終了処理で該当領域を umount する．umount 処理がエラー返却されるとジョブ運用ソフトウェアで問題が発生するため，現状ではランク番号ディレクトリを使用しないジョブを対象としている．また，本提案手法では，同一ディレクトリへのファイル作成を行うジョブを対象としているが，同一ファイルへの truncate 処理や書き込み操作など，その他長期化する I/O リクエストのパターンに対応した処理中断機能の検討を行う予定である．今後は対応できるジョブのパターンを増やすとともに，継続してファイルシステムの運用改善活動に取り組む．

謝辞 本稿の結果は，理化学研究所の「京」を利用して得られたものである．技術的な支援を頂いた富士通株式会社の宮本巧輝氏に感謝する．

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] Martino, C. D., Kalbarczyk, Z., Iyer, R. K., Baccanico, F., Fullop, J. and Kramer, W.: Lessons Learned from the Analysis of System Failures at Petascale: The Case of Blue Waters, *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, IEEE, pp. 610–621 (2014).
[4] Shoji, F., Matsui, S., Okamoto, M., Sueyasu, F., Tsukamoto, T. and Uno, A.: Long term failure analysis of 10 Petascale supercomputer, *HPC in Asia Session at ISC* (2015).
[5] 辻田祐一，義崎竜彦，山本啓二，末安史親，宮崎亮二，宇野篤也：「京」のファイルシステムの運用改善への取り組み，情報処理学会研究報告，Vol. 2016-HPC-156, No. 12, pp. 1–10 (2016).
[6] 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. 2017-HPC-162, No. 12, pp. 1–8 (2017).
[7] 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).
[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] Spitz, C. and Koehler, A.: Tips and Tricks for Diagnosing Lustre Problems on Cray Systems, *2011 Cray User Group Meeting (CUG)* (2011).
[10] Ezell, M., Mohr, R., Wynkoop, J. and Braby, R.: Lustre at Petascale: Experiences in Troubleshooting and Upgrading, *2012 Cray User Group Meeting (CUG)* (2012).
[11] Saini, S., Rappleye, J., Chang, J., Barker, D., Mehrotra, P. and Biswas, R.: I/O Performance Characterization of Lustre and NASA Applications on Pleiades, *High Performance Computing (HiPC), 2012 19th International Conference on*, pp. 1–10 (2012).
[12] Du, M., Li, F., Zheng, G. and Srikumar, V.: DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS'17, ACM, pp. 1285–1298 (2017).
[13] Das, A., Mueller, F., Hargrove, P., Roman, E. and Baden, S.: Doomsday: Predicting Which Node Will Fail When on Supercomputers, *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC'18, No. 9, IEEE Press (2018).

© 2019 Information Processing Society of Japan