Vol.2013-HPC-139 No.11
2013/5/29

ページ回収に伴う OS ノイズの除去

石黒　駿[†]　村上　じゅん[†]　大山恵弘[†,††]

OS ノイズは，高性能計算アプリケーションの性能低下の大きな要因の一つである．OS ノイズの実体は，割り込み処理などの OS カーネルによるサービスの実行や，メモリ管理デーモンなどの各種デーモンの実行である．これらの動作は，アプリケーションによる計算を中断させ，計算時間を増大させる．OS ノイズは，多くのプロセスやスレッドが頻繁に同期するアプリケーションで大きく性能を低下させる．本論文では，メモリページの回収処理に伴う OS ノイズに着目し，その OS ノイズによる影響を低減する手法を提案する．Linux のディスク I/O では，通常はファイルデータはメモリ上にキャッシュされる．大量のディスク I/O を行った結果，キャッシュ用のメモリが足りなくなると，OS は特別なカーネルスレッドを起動し，近い将来に利用されないと思われるページを回収させる．このカーネルスレッドが頻繁に動作すると，その OS ノイズによりアプリケーションの性能が低下する．提案手法は，大量のディスク I/O が行われている場合に，ページ回収のカーネルスレッドに先行して，さらに大きな単位でページを回収する．これによりページ回収の回数が減り，OS ノイズによる影響が小さくなる．我々は提案手法に基づくシステムを実装し，実験を行った．その結果，その OS ノイズによる性能低下をほぼなくすことに成功した．

Elimination of OS Noises Caused by Page Reclamation

Shun Ishiguro ,[†] Jun Murakami [†] and Yoshihiro Oyama[†,††]

OS noises are one of the major causes of performance degradation in applications of high performance computing. OS noises are execution of services by the operating system kernel such as interrupt handling or execution of various daemons such as a memory management daemon. These execution interrupts the computation of an application and increases the execution time. OS noises significantly degrade the performance of an application in which many processes or threads frequently synchronize with each other. In this paper, we focus on a OS noise caused by reclamation of memory pages and propose a method of reducing the effect of the OS noise. The disk I/O of Linux usually caches file data on memory. When numerous disk I/O occur and the OS runs out of memory for caching file data, the OS activates a special kernel thread that reclaims memory pages that are unlikely to be used in the near future. If the kernel thread is frequently activated, the performance of an application is degraded due to its OS noise. The proposed method reclaims memory pages in advance of the kernel thread for page reclamation. It reclaims more pages than the kernel thread, and thus reducing the frequency of page reclamation and the effect of the OS noise. We implemented a system based on the proposed method and conducted an experiment. Results of the experiment showed that the proposed method could almost eliminate the performance degradation caused by the OS noise.

1. はじめに

スパコンなどを用いる高性能計算の場面においては，OS ノイズによる性能低下を最小化することが極めて重要である．OS ノイズとは，割り込みハンドラやタスクレットなどの OS カーネルによる処理や OS サービスを提供するデーモンによる処理であり，アプリケーションの計算性能を低下させる．たとえば，OS カーネルはタイマ割り込みやネットワーク通信が生じたら，それに対処する処理を実行しなければならない．また，定期的にデーモンをスケジューリングする必要があることもある．割り込み処理の実行やデーモンの動作の際には，アプリケーションを実行しているプロセスやスレッドは一時的に CPU を取り上げられる．頻繁に同期を行うアプリケーションでは，一部のプロセスやスレッドにおける計算の遅れが，計算全体の性能に大きな影響を与える．多くの OS ノイズはアプリケーションの処理とは独立に発生するため，同じ計算

† 電気通信大学
The University of Electro-Communications
†† 独立行政法人科学技術振興機構, CREST
JST, CREST

ⓒ 2013 Information Processing Society of Japan

を行うプロセスやスレッドが計算を終えるまでの時間は同じになるとは限らない．その結果，バリア同期を取るようなアプリケーションでは，計算の全体の実行時間は，同期地点に到達するのが一番遅いプロセスやスレッドの実行時間にまで伸びる（**図 1**）．OS ノイズがアプリケーションの性能を低下させる問題は，複数の研究により報告されている [2,4,6–8,12]．

OS ノイズには多くの種類があるが，本研究では特に，大規模なデータのディスク I/O を実行するアプリケーションにおいて生じる OS ノイズを考える．本研究で行った実験によると，ディスク I/O が原因で OS が実行する空きメモリ確保処理によって OS ノイズが生じ，それがアプリケーションの性能を低下させることがわかった．そこで，本論文では，そのような OS ノイズによる影響を調査した結果を報告し，その影響を縮小する手法を提案する．高性能計算の分野では Linux が圧倒的に多く利用されているため，本研究でも OS は Linux であると仮定する．

OS は空きメモリにファイルデータを置き，キャッシュとして利用する．しかし，大きなファイルデータを読み込むと，キャッシュのためのメモリが足りなくなる．すると OS は，キャッシュの一部を解放して空きメモリを増やし，そのメモリを，将来に読み込むデータのためのメモリ領域として再利用する．ファイルを読み続けると，この空きメモリ確保のための処理が頻繁に発生する．これは一度に解放するキャッシュの量が少ないためである．本研究では，一度に解放するキャッシュメモリの量を大きくし，空きメモリ確保処理の頻発を防ぐ．

上記の OS ノイズは，大規模なファイルデータのディスク I/O を実行した時に生じやすい．一般に，高性能計算アプリケーションのためのストレージには，大容量，高性能，高可用性が要求される．よって，それらの要求を満たすべく作られた並列分散ファイルシステムが利用されることが多い [5,9–11,13,16]．並列分散ファイルシステムを用いると，アプリケーションが行う I/O とは別に，並列分散ファイルシステムを構成するデーモンなどが I/O を行う．並列分散ファイルシステムは，複数の I/O ノードにデータを分散して配置する．I/O ノードと計算ノードは，同一ノードである場合もあれば，別ノードである場合もある．同一ノードの場合，ローカルディスクに対する I/O が行われる．その際に読み書きされるデータは，OS によってそのノードのメモリにキャッシュとして配置される．別ノードの場合でも，ファイルシステムによってはファイルデータがメモリにキャッシュとして配置

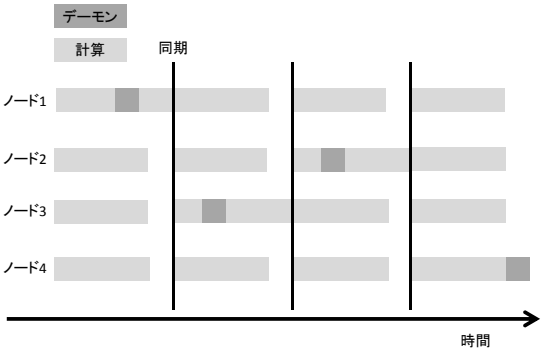

図 1 OS ノイズによるアプリケーションの実行への影響
Fig. 1 Effect of an OS noise on execution of an application

されることがある．その結果，ディスク I/O により，計算そのものに利用できるメモリが圧迫される．

本論文の構成は以下の通りである．2 章では OS ノイズによる問題について述べる．3 章では提案手法の設計と実装について説明し，4 章でそれを議論する．5 章で提案手法を評価した実験の結果を報告する．6 章では関連研究について述べる．7 章で本研究のまとめと今後の課題を述べる．

2. ページ回収による OS ノイズ

2.1 Linux におけるメモリ管理

Linux では，空きメモリが不足するとカーネルスレッド kswapd がページを回収する．ページの回収は，ページキャッシュの解放やスワップアウト処理により実現する．OS は定期的に空きメモリ容量をチェックしており，その値が閾値を下回ると kswapd を動作させる．kswapd は，空きメモリがある程度の容量になるまでページを回収する．

Linux でファイルを読み込むと，通常はその内容がページキャッシュとしてメモリに置かれる．ここで，物理メモリサイズを超えるような大容量のファイルを読み込む処理を頻繁に実行するプログラムを考える．この実行では，多くのメモリがページキャッシュのために消費され，空きメモリが不足する．すると kswapd が，アクセス頻度の低いページ（ページキャッシュ含む）を解放する．しかし，読み込み処理が頻繁に実行されている限り，空いたページにはすぐにファイルデータがページキャッシュとして置かれる．その結果，kswapd はページ回収処理を頻繁に実行することになる．kswapd はアプリケーションから一時的に CPU を奪うため，kswapd による OS ノイズがアプリケーションの実行時間を増大させる．

Linux でのページ回収は，プロセスやカーネルにペー

ⓒ 2013 Information Processing Society of Japan

ジを割り当てる時と，空きページ数が一定以下になった時に実行される．例えば，カーネルがページキャッシュ用のメモリを要求した時に，空きページが足りなければページが回収される．また，空きページの容量が閾値 `/proc/sys/vm/min_free_kbytes` を下回ると，ページが回収される．なお，`min_free_kbytes` に値を書き込むことで，閾値を変更可能である．kswapd は空きページの容量が一定値を越えるまでページを回収する．この容量は，`min_free_kbytes` の値をもとに算出される値である．

kswapd はカーネルスレッドであり，NUMA 環境では複数スレッド存在する．NUMA 環境で動作する Linux では，CPU とメモリのセットごとにメモリを管理している．NUMA 環境では kswapd は複数存在するが，それぞれの kswapd が管理するメモリはセット単位となっている．よって，CPU とメモリのセットが 2 セットある NUMA 環境では kswapd は 2 つ動作する．kswapd は，自分が担当するメモリを検査し，解放可能なページがあれば，それを解放する．

2.2 高性能計算における OS ノイズ

このページ回収はファイルの読み込みや書き込みに伴い頻繁に発生する．高性能計算のアプリケーションでは，計算用の初期データを読み込んだり，計算結果を書き出したりする．計算ノードのローカルファイルシステムを用いる場合や，並列分散ファイルシステムの I/O サーバが計算ノード上で動作している場合に，ページキャッシュに多くのメモリが使われる．その結果，ページ回収の処理が OS ノイズとしてアプリケーションに影響を与える可能性がある．

並列分散ファイルシステムの I/O サーバとアプリケーションを同一ノード上で動作させることを想定に入れたファイルシステムは複数存在する [5,10,11]．I/O サーバは，遠隔の計算ノードからデータの読み書き要求があれば，そのデータの I/O を行う．よって，この I/O が OS ノイズとなり，そのノードで動くアプリケーションの実行時間を増大させる．アプリケーションがアクセスするデータを，計算ノードと異なるノード上にある I/O サーバが提供している場合でも，アプリケーションが計算とファイル入出力をオーバラップさせる処理をしている場合には，OS ノイズの影響が生じる可能性がある．

2.3 実験結果

kswapd のページ回収処理による OS ノイズが与える影響を調べるため，実験を行った．詳細は 5 章で述べるが，ここで結果の概要だけを述べる．実験では，気象予測アプリケーションである WRF 3.4 [14] と，物理メモリサイズを超えるサイズのファイルを繰り返し読み込むプログラム（ノイズプログラム）を同時に実行した．ノイズプログラムの実行により，kswapd が頻繁に動作する．この OS ノイズによる WRF の実行時間の変化を調査した．

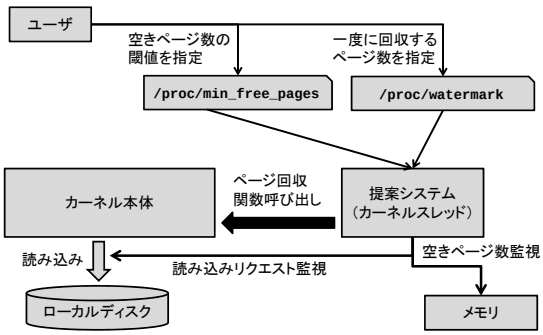

図 2 提案システムの構成
Fig. 2 Structure of the proposed system

WRF の実行時間は，ノイズプログラムを実行しない場合は 5735 秒だったが，実行した場合には 6542 秒だった．すなわち，ノイズプログラムの実行により，WRF の実行時間が約 14%長くなった．WRF はノード間で同期を取るが，ノイズプログラムを実行しているノードの計算がページ回収による OS ノイズの影響で遅くなり，その結果，全体の計算時間が長くなったと考えられる．

3. 提案手法

3.1 概　　要

提案手法では，ファイルの読み込みがアプリケーションに影響を与える場合を想定し，kswapd の頻繁なページ回収による影響を削減するために，一度に大量のページを回収する．提案手法に基づくシステムの構成を**図 2** に示す．提案システムは，Loadable Kernel Module (LKM) を用いてカーネルスレッドとして実装した．このカーネルスレッドはローカルディスクに発行される読み込みリクエスト数と空きページ数を監視する．それらの情報をもとに，ページを kswapd によらずに自身で回収する．

読み込みリクエスト数が多い場合は，大きいデータがローカルディスクから読み込まれてページキャッシュとしてメモリを消費している可能性が高い．そこで，空きページ数が一定以下になった際に，それが大きいファイルデータの読み込みによるものであると判断したら，ページを回収するカーネル関数を呼び出す．提案システムによるページ回収を図示したものを**図 3** に示す．ここでの空きページ閾値は kswapd が動作す

ⓒ 2013 Information Processing Society of Japan

る値である min_free_kbytes より大きくする．これにより，kswapd によるページ回収が始まる前に提案システムがページを回収し，kswapd を動作させずに済む．

一方，空きページ数が一定以下になった場合でも，それが大きいファイルデータの読み込みによるものではないと判断すれば，ページ回収は行わない．この場合は，その後に空きページサイズが min_free_kbytes を下回れば kswapd が動作し，通常通りページキャッシュの解放やスワップアウトが行われる．このように判断することにより，アプリケーションがメモリを物理メモリサイズの上限近くまで活用できる．空きページ数のみでページを回収するかどうかを判断すると，アプリケーションの実行そのものが原因で空きページ数が閾値を下回った場合に，提案システムによるページ回収が頻繁に発生してしまう．

3.2 実装の詳細

ページ回収の契機となる空きページ数の閾値と一度に回収するページ数は，ユーザが指定できるようにした．それぞれ /proc/min_free_pages と /proc/watermark に，ユーザがページ数を書き込むことで設定できるようにした．

Linux のディスクに関する情報は /proc/diskstats から取得できる．ここには，読み込み完了リクエスト数，書き込み関数リクエスト数，実行中の I/O リクエスト数，読み込んだセクタ数，読み込みにかかった時間，書き込んだセクタ数，書き込みにかかった時間などが記録されている．このファイルにより読み込みリクエスト数を定期的に監視し，短時間に多くの読み込みが行われた場合は，大きいファイルデータを読み込んでいると判断する．提案システムでは，1 秒ごとに読み込みリクエスト数を監視している．本研究の実験環境においては，大きなファイルを読み込んだ場合の 1 秒あたりのリクエスト数は 1,500 程度となるため，大きなファイルデータを読み込んでいると判断するリクエスト数の閾値を 1,500 とした．

空きページ数は，カーネルの global_page_state マクロを利用することで取得できる．提案システムでは 1 秒ごとに global_page_state マクロによって空きページ数を取得し，閾値を下回っていないかを検査する．

提案システムがページ回収のために呼び出しているカーネル関数は do_try_to_free_pages である．この関数は，回収したいページ数などを指定した構造体を渡すと，ページを回収し，回収したページ数を返す．この関数は，kswapd やメモリ割り当て時のページ回収でも呼び出される．

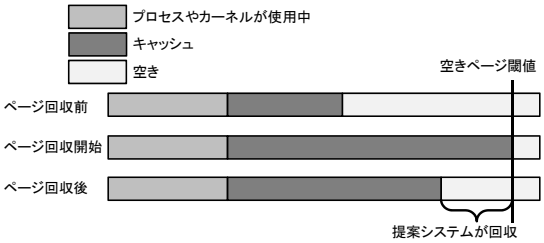

図 3 提案システムによるページ回収
Fig. 3 Page reclamation by the proposed system

4. 議　　論

4.1 ページキャッシュを解放する方法

ページキャッシュを解放するには，元々，Linux が /proc/sys/vm/drop_caches を利用する方法を提供している．このファイルに 1 を書き込むとページキャッシュがすべて解放される．しかし，この方法ではすべてのページキャッシュが解放されるため，頻繁にアクセスするファイルデータについても再読み込みが必要となるという問題がある．

4.2 Direct I/O の利用

ファイルを読み込む際に Direct I/O を用いるとページキャッシュを使わないため，ページ回収に伴う OS ノイズが発生しない．Direct I/O は，ページキャッシュを介さず，ディスクとアプリケーションが直接データをやりとりする I/O である．しかし，Direct I/O の利用は逆に性能を低下させることがある．Direct I/O の利用は独自にキャッシュ機構を持つプログラムを前提としており，例えば一部のデータベースソフトウェアなどが利用する．キャッシュ機構を持たないアプリケーションの実行では，一度読み込んだファイルを再度読むなどの場合に，性能が低下する．また，Linux にはファイルデータを先読みしてページキャッシュに載せる機能があるが，ページキャッシュを使わないと先読みができなくなり，性能が低下することがある．さらに，Direct I/O を用いるにはアプリケーションの修正が必要となるという問題もある．

提案手法は，OS カーネル内のみで処理が完結するため，アプリケーションを修正する必要はない．さらに，ページキャッシュを利用するので，先読みによる読み込み性能の向上が期待できる．

4.3 OS ノイズの一般的な削減法

OS ノイズの一般的な削減法として，デーモン専用コアを設ける方法がある．これは，ノード内のコアのうち 1 つだけをデーモン専用とし，残りをアプリケー

ⓒ 2013 Information Processing Society of Japan

ションに割り当てるものである．これによって，デーモンがアプリケーションから CPU を奪うことがなくなる．しかし，kswapd は NUMA 環境ではノード数だけ存在する．さらにページ回収が頻発する状態では，kswapd が多くの CPU 時間を必要とする．そのため，ノード数が増えた場合に 1 つのコアでは処理が追いつかなくなる可能性がある．また，専用コアを kswapd の数だけ割り当てると，ページ回収が並列に実行できて効率が上がるが，今度はアプリケーションに使えるコアが減る．また，kswapd が消費する CPU 時間自体は変わらない．提案手法では，一度に大量のページを回収するので，頻繁なページ回収は発生せず，kswapd によるページ回収と比べて CPU 時間の消費は少ない．

5. 実　　験

5.1 実験手法

提案システムを利用する時と利用しない時とで，アプリケーションの性能を比較する実験を行った．

実験環境を**表 1** に示す．表に示すノードを 4 台準備し，それらを InfiniBand QDR でスイッチを介して接続した．なお，MPI でアプリケーションを並列に実行する際に実行ファイルを共有する必要があるが，本実験では，そのために NFS を用いた．

実験では，気象シミュレーションモデルを実装したアプリケーションである WRF 3.4 [14] を用いた．WRF は一定のシミュレーション時間刻みで，気象をシミュレーションする．実験では 6 秒刻みでシミュレーションした．例えば，シミュレーション時刻が 12 時 00 分 00 秒からシミュレーションを開始すれば，12 時 00 分 06 秒，12 時 00 分 12 秒，12 時 00 分 18 秒といった時点での結果が出力される．この一つ一つの計算の刻みをステップという．計算結果はメモリ上にあり，次のステップで更新されるが，更新前にスナップショットのようにファイルに書き出すこともできる．実験では，ページ回収による OS ノイズが計算に与える影響を調べるため，WRF によるファイルの保存は行わなかった．

以下の 3 つの場合で WRF の実行時間を測定した．

Original: OS ノイズを生じさせるプログラムを走らせない場合

Noise: OS ノイズを生じさせるプログラムを走らせ，提案システムを用いない場合

Noise+Proposed: OS ノイズを生じさせるプログラムを走らせ，かつ，提案システムを用いる場合

実験は，物理メモリサイズを超えるサイズのファイルを読み込み続けるプログラム（ノイズプログラム）と WRF を同時に実行し，WRF の実行時間を計測するものである．ノイズプログラムによりページキャッシュが消費されるため，ページ回収が頻繁に発生することが予想される．

表 1 実験に用いたノードとソフトウェアの仕様
Table 1 Specification of computing nodes and software used in experiments

CPU	Intel Xeon E5645 2.4 GHz (6 core) × 2
memory	48 GB
HDD	SAS 15,000 rpm 600 GB
OS	CentOS 6.3 64 bit (kernel 2.6.32)
file system	ext4
MPI	MPICH2

ノード内での実験とノード間での実験の両方を行った．ノード内での実験では，マルチスレッドで WRF を実行した．ノイズプログラムは同一ノード上で実行した．ノード間の実験では，4 ノードを用いた．ノード内ではマルチスレッド，ノード間では MPI によるマルチプロセスで WRF を並列に実行した．ノイズプログラムは，4 ノード中 1 ノードのみで実行した．各ノードで生成する WRF のスレッド数は，物理コア数より 1 つ少ない 11 とした．どのノードのスレッド数も 11 とした理由は，ノイズプログラムを含む他のプログラムの実行が CPU を使うことによる影響を最小化するためと，全ノードで条件を揃えるためである．

提案システムのパラメタとして，空きページ数の閾値に 262144（1GB），一度に解放するページ数に 1048576（4GB）を用いた．

5.2 実験結果

1 ノードのみで実験した時の，WRF の 1 ステップの計算時間と累積計算時間を**図 4** と**図 5** に，4 ノードを用いて実験した時の WRF の 1 ステップの計算時間と累積計算時間を**図 6** と**図 7** に示す．

1 ノードでの実験では，WRF のみを実行した場合に比べて，ノイズの影響を受けた場合は，1 ステップの計算時間が長くなる傾向があった．WRF のみの場合，1 ステップの計算時間はほぼ一定であるが，ノイズプログラムを実行すると計算時間がばらついた．一方で提案システムを用いた場合，ノイズプログラムが動いていても，計算時間はほぼ一定であり，WRF のみを実行した場合とほぼ同じ程度になった．累積計算時間は，ノイズの影響を受けた場合には，ノイズの影響を受けない場合と比較して約 9%長くなった．提案システムを用いると，累積計算時間はノイズの影響を受けない場合とほぼ同じになった．

4 ノードの実験でも，WRF のみを実行した場合と比較して，ノイズプログラムを実行した場合に 1 ステッ

ⓒ 2013 Information Processing Society of Japan

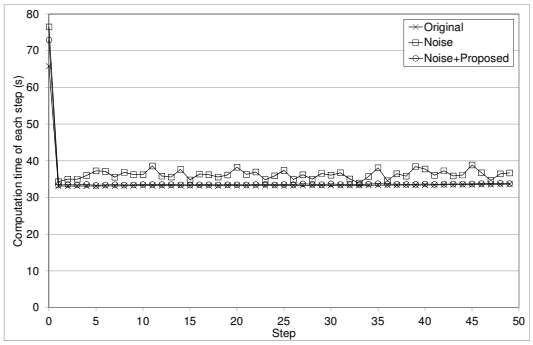

図 4 1 ノードでの 1 ステップの計算時間
Fig. 4 Computation time of one step in execution on one node

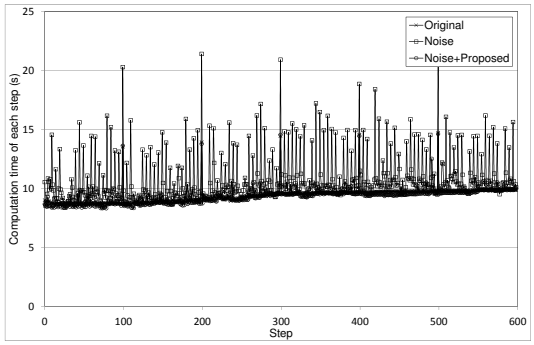

図 6 4 ノードでの 1 ステップの計算時間
Fig. 6 Computation time of one step in execution on four nodes

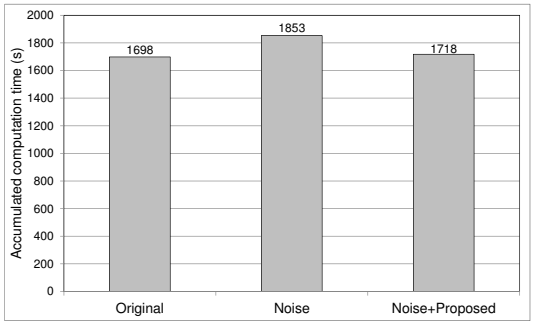

図 5 1 ノードでの累積計算時間
Fig. 5 Accumulated computation time in execution on one node

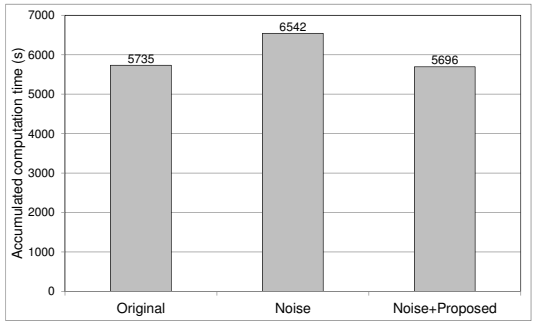

図 7 4 ノードでの累積計算時間
Fig. 7 Accumulated computation time in execution on four nodes

プの計算時間が長くなった．図 6 では，線が重なっているためわかりにくくなっているが，提案システムを用いた場合の折れ線は，ノイズプログラムを実行しない場合の折れ線と非常に近い場所にある．すなわち，各ステップの実行時間はほぼ同程度だった．累積計算時間についても 1 ノードの実験と同じく，ノイズプログラムを実行しない場合と提案システムを利用した場合でほぼ同じになった．ノイズの影響を受ける場合には，受けない場合と比較して累積計算時間は約 14%増加した．

1 ノード上での実験結果と 4 ノード上での実験結果を比較すると，4 ノード上での実験のほうが，ノイズの影響が大きいことがわかった．図 6 を見ると，ステップによっては，ノイズプログラムの実行により，実行させない場合の倍程度にまで実行時間が伸びることもあることがわかる．

5.3 考　　察

提案手法の導入により，ノイズプログラムが走っていてもアプリケーションの実行時間がほとんど変化しなくなった．これは，kswapd が呼び出されなくなったためであると考えられる．提案手法を用いた場合でも，ページ回収のためのカーネルスレッドがアプリケーションから CPU を奪うことは依然としてある．しかし，その頻度が kswapd よりも低いため，影響が小さく抑えられている．

一度に解放するページ数を大きくとると，その分だけページ回収にかかる時間は長くなる．しかし，一度に少量のページを解放する処理を頻繁に繰り返すよりも，一度に多くのページを解放するコストの方が小さいことを，実験結果は示唆している．この原因としては，kswapd はプロセスのスワップアウトなど，キャッシュの解放以外の処理も担当しているので，提案システムのカーネルスレッドよりも処理が複雑であることが考えられる．もう一つの原因として，コンテキストスイッチのオーバーヘッドが考えられる．kswapd が頻繁に起動するとコンテキストスイッチも頻繁に発生する．そのため，そのオーバーヘッドがアプリケーションの実行時間への影響に含まれたと考えられる．提案手法では，kswapd と比べるとカーネルスレッドは頻繁には起動しないため，その分だけコンテキスト

ⓒ 2013 Information Processing Society of Japan

スイッチが実行される回数が少なくなる．

なお，閾値に関しては，アプリケーションの特性，メモリの搭載量，ページサイズなどによって最適な値が異なると考えられる．WRF では動的なメモリ確保は比較的緩やかなペースである．しかし，アプリケーションによってはメモリの解放や確保を頻繁に行う．今回の実験結果と似た性能傾向が出やすいのは，アプリケーションによるメモリ使用量がある程度固定されている状況であると考えられる．

6. 関連研究

青木らの研究 [15] では，Linux 上でのリアルタイムタスクの動作を保証するために，ページ回収専用のコアを設ける手法を提案している．この手法では，マルチコア CPU を活用し，アプリケーションの専用コアとページを回収するコアを分離している．通常 NUMA 環境ではノードごとに kswapd が存在するが，この研究による実装では，コアごとに kswapd が存在するように改造されている．さらに，アプリケーションを実行するコアでは空きページ数の閾値を小さくし，ページを回収するコアでは閾値を大きくすることで，ページ回収を実行するコアを固定している．リアルタイムアプリケーションでは，メモリ確保の遅延が問題となる．そこでアプリケーションとは別のコアで事前にページを回収しておくことで，メモリ要求があった際に即座にメモリの割り当てができる．本研究では，ページ回収の実行については kswapd を利用しない点で異なる．また，ファイルの読み込みによるページキャッシュ使用がページ回収処理頻発の原因となることに言及している点では本研究と同じであるが，青木らの研究ではファイル読み込みを監視していない．本研究では，ファイルの読み込みによりメモリが不足した場合のみ，ページを回収するという点で異なる．また，対象となるアプリケーションは，組み込み分野のようなリアルタイムアプリケーションではなく，並列計算アプリケーションである．

HPC クラスタ向けの OS として CAOS [1] がある．CAOS では複数のノードの動作を協調させることで OS ノイズを削減する．OS ノイズの主な原因にタイマ割り込みがある．CAOS では全ノードで共通のタイマを利用することで，タイマ割り込み処理を同期している．共通のタイマとして，マスターノードが存在し，マスターノードが全ノードに一斉にハートビートを送信する．CAOS はハートビートを受け取ると，ネットワーク割り込み処理を経由して，最終的にタイマ割り込み処理を実行する．これにより，タイマ割り込み処理が全ノードで同期されるため，OS ノイズとしての影響を削減できる．本研究では，割り込み処理ではなく，デーモンの動作を対象としている．kswapd の場合，不定期に実行される．また全ノードで同時に動作を開始しても，終わるタイミングがノードごとに異なる．したがって，タイマ割り込みを同期するだけでは，ページ回収によるノイズを削減することは難しい．

De らの研究 [3] では，複数アプローチを用いて OS ノイズを削減している．スケジューリングに伴うノイズについては，アプリケーションをリアルタイムプロセスとしてラウンドロビンでスケジューリングすることで，CPU を奪われにくくしている．また，Linux ではワークキューにタスクレットなどのカーネルの処理が登録されており，プロセスと同様スケジューリングされて実行される．このワークキューは CPU コアごとにある．アプリケーションが実行されているコアでは，ワークキューを配置しないことで，アプリケーションの実行を阻害しないようにしている．外部割り込みについても，アプリケーションを実行しているコアで割り込み処理を行わず，専用のコアで割り込みを処理するようにしている．また，SMT の優先度をアプリケーションが変更できるようにしている．Linux では，プロセスが SMT のスレッドの優先度を指定しても，割り込みコンテキストからの復帰時に優先度をリセットしてしまうという問題があった．そこで，優先度を指定するシステムコールを用意することで，ユーザが任意に優先度を指定できるようにしている．本研究では，デーモンが動作する頻度を少なくしている．コアを固定するだけでは，デーモンが動作する頻度は変わらず，消費する CPU 時間も減らない．

7. おわりに

本論文では，ページ回収処理がアプリケーションの実行時間に与える影響を削減する手法を提案し，その評価を報告した．提案手法は，ページキャッシュに使うメモリを確保するためのページ回収処理の頻度を，一度に回収するページ数を大きくすることにより削減する．ページ回収を行うかどうかは，空きページ数とファイル読み込みリクエスト数を元に判断する．実験ではページ回収に伴う OS ノイズにより，アプリケーションの実行時間が最大 14%ほど長くなった．しかし，提案手法によりその影響をほぼなくすことができた．

今後の課題を以下で述べる．第一に，ファイルの書き込みへの対応が挙げられる．ファイルの書き込みも，読み込みと同様にアプリケーションの実行時間に影響を与える．Linux では，ファイルに書き込む時は，ま

ⓒ 2013 Information Processing Society of Japan

ずページキャッシュにデータを書き込む．データを書き込んだページは dirty なページとしてマークしておき，定期的にディスクへ書き出す．ファイルの書き込みが多く行われている状況でページを回収する場合，dirty なページをディスクへ書き出す処理が多く発生する．しかし，ディスクへのデータ書き出しには時間がかかる．ゆえに，ファイルを読み込む状況でページを回収する場合のように，すぐにページを回収できるとは限らない．ページ回収処理だけでなく，ディスクへの書き出しを考慮した手法を考える必要がある．第二は，遠隔の I/O ノードに存在するファイルへの対応である．現在の実装ではディスク負荷情報を用いて，大きなファイルデータが読み込まれているかどうかを判断している．この手法では，ローカルディスク上のファイルへのアクセス負荷は把握できるが，リモートの I/O ノード上のファイルへのアクセス負荷は把握できない．リモートのファイルの読み込みにおいても，並列分散ファイルシステムがページキャッシュを利用する場合には，OS ノイズにより性能が低下する可能性がある．よって，リモートのファイルへのアクセスも把握する必要がある．

謝　　辞

本研究を行うにあたり，貴重で有益な助言をいただいた筑波大学の建部修見准教授に深く感謝する．本研究は科学技術振興機構戦略的創造研究推進事業（JST CREST）の研究課題「ポストペタスケールデータインテンシブサイエンスのためのシステムソフトウェア」の支援を受けている．

参 考 文 献

1) Betti, E., Cesati, M., Gioiosa, R. and Piermaria, F.: A Global Operating System for HPC Clusters, *Proceedings of the 2009 IEEE International Conference on Cluster Computing* (2009).
2) De, P., Kothari, R. and Mann, V.: Identifying Sources of Operating System Jitter Through Fine-Grained Kernel Instrumentation, *Proceedings of the 2007 IEEE International Conference on Cluster Computing*, pp. 331–340 (2007).
3) De, P., Mann, V. and Mittaly, U.: Handling OS Jitter on Multicore Multithreaded Systems, *Proceedings of the 23rd IEEE International Symposium on Parallel and Distributed Processing* (2009).
4) Ferreira, K. B., Bridges, P. and Brightwell, R.: Characterizing Application Sensitivity to OS Interference Using Kernel-Level Noise Injection, *Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (2008).
5) GlusterFS, `http://www.gluster.org/`.
6) Hoefler, T., Schneider, T. and Lumsdaine, A.: Characterizing the Influence of System Noise on Large-Scale Applications by Simulation, *Proceedings of SC10* (2010).
7) Jones, T.: Linux Kernel Co-Scheduling For Bulk Synchronous Parallel Applications, *Proceedings of the 1st International Workshop on Runtime and Operating Systems for Supercomputers*, pp. 57–64 (2011).
8) Morari, A., Gioiosa, R., Wisniewski, R. W., Cazorla, F. J. and Valero, M.: A Quantitative Analysis of OS Noise, *Proceedings of the 2011 IEEE International Parallel and Distributed Processing Symposium*, pp. 852–863 (2011).
9) Schwan, P.: Lustre: Building a File System for 1,000-node Clusters, *Proceedings of the 2003 Linux Symposium* (2003).
10) Shvachko, K., Kuang, H., Radia, S. and Chansler, R.: The Hadoop Distributed File System, *Proceedings of the 26th IEEE Symposium on Massive Storage Systems and Technologies* (2010).
11) Tatebe, O., Hiraga, K. and Soda, N.: Gfarm Grid File System, *New Generation Computing*, Vol. 28, No. 3, pp. 257–275 (2010).
12) Tsafrir, D., Etsion, Y., Feitelson, D. G. and Kirkpatrick, S.: System Noise, OS Clock Ticks, and Fine-Grained Parallel Applications, *Proceedings of the 19th ACM International Conference on Supercomputing*, pp.303–312 (2005).
13) Weil, S., Brandt, S. A., Miller, E. L., Long, D. D. E. and Maltzahn, C.: Ceph: A Scalable, High-Performance Distributed File System, *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, pp. 307–320 (2006).
14) WRF, `http://www.wrf-model.org/`.
15) 青木英郎, 長井昭裕, 関山友輝, 大島訓: コア別ページ回収によるマルチコアシステムの安定性改善, 情報処理学会研究報告, Vol. 2012-OS-122, No. 18 (2012).
16) 宇野篤也, 加藤丈治, 宮本巧輝, 岩田章孝, 長屋忠男: スーパーコンピュータ「京」: 4. システムソフトウェア-OS，運用管理ソフトウェア，ファイルシステム-, 情報処理, Vol. 53, No. 8, pp. 774–779 (2012).

ⓒ 2013 Information Processing Society of Japan