Vol.2012-ARC-200 No.9
Vol.2012-OS-121 No.9
2012/5/7

Linuxのメモリ管理に起因する性能問題と回避策の評価

及川 一樹[1,a)] 佐藤 孝治[1,b)] 犬束 敏信[1,c)] 山本 公洋[1,d)] 鷲坂 光一[1,e)] 冨田 清次[1,f)]

概要：本稿では，Linux のメモリ管理に起因するアプリケーションの性能劣化事象，およびその原因分析と回避策について報告する．筆者らは Linux 上で動作する大規模分散処理システムの開発を進める過程で，その構成要素である分散ファイルシステムの性能劣化という事象に遭遇した．本事象を詳細に分析する中で，性能劣化が Linux カーネルのメモリ管理サブシステム内でのロック競合に起因することを突き止めたので，これを詳細に報告する．本事象については，最新の Linux カーネルを用いても改善されないケースが有ること，および，カーネルの変更ができないシステムも存在することを想定し，呼び出すシステムコールの種類や呼び出し方法を工夫するなど，アプリケーションレベルでの修正による回避策について検討，評価を行った．これらの回避策とその効果の定量的な評価についても併せて報告する．

キーワード：Linux, 仮想メモリ管理, マルチスレッド, セマフォ, 性能問題

Performance problem in Linux kernel memory management subsystem

Oikawa Kazuki[1,a)] Sato Koji[1,b)] Inuzuka Toshinobu[1,c)] Yamamoto Kimihiro[1,d)]
Washisaka Mitsukazu[1,e)] Tomita Seiji[1,f)]

Abstract: In this paper, we report the performance problem in Linux kernel memory management subsystem, and propose its workarounds. While we had been developing a distributed file system over Linux OS as a component of a large scale distributed data processing system, we encountered the performance problem On the investigation, we found out this problem was due to the lock exclusive control in the memory management subsystem of Linux kernel. In this paper, we propose the workarounds such as using another system call and changing the way to call the system call, and report the experimental results to show its effectiveness.

Keywords: Linux, Virtual Memory Management, Multithread, Semaphore, Performance Problem

1. はじめに

筆者らは Linux 上で動作する大規模分散処理システムの開発を進めている．その過程で構成要素である分散ファイルシステムの性能劣化という事象に遭遇した．本事象を詳細に分析する中で，性能劣化が Linux のメモリ管理に起因することを突き止めたので，これを詳細に報告する．

本事象については，最新の Linux カーネルを用いても改善されないケースが有ること，および，カーネルの変更ができないシステムも存在することを想定し，アプリケーションの修正による回避策について検討，評価を行った．これらの回避策とその効果の定量的な評価についても併せて報告する．

[1] NTT ソフトウェアイノベーションセンタ
NTT Software Innovation Center
a) oikawa.kazuki@lab.ntt.co.jp
b) sato.koji@lab.ntt.co.jp
c) inuzuka.toshinobu@lab.ntt.co.jp
d) yamamoto.kimihiro@lab.ntt.co.jp
e) wasisaka.mitukazu@lab.ntt.co.jp
f) tomita.seiji@lab.ntt.co.jp

2. 背景

我々は，大量のデータを蓄積・分析するための大規模分散処理基盤 CBoC タイプ 2[1] を開発している．CBoC タイプ 2 は，大量のデータを多数のサーバに分散格納する GFS[2]

© 2012 Information Processing Society of Japan

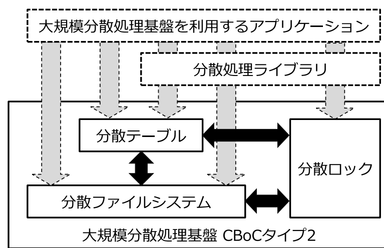

図 1 CBoC タイプ 2 の構成

や HDFS[3] と類似の分散ファイルシステム，大量のデータを構造化データとして管理する BigTable[4] や HBase[5] と類似の分散テーブルシステム，そしてこれらの分散システムの可用性を高めるための基本機能を提供する分散ロックという 3 つの分散処理のサブシステムで構成されている (図 1).

分散ファイルシステムでは，ファイルをチャンクと呼ばれる，64MB 程度の固定長サイズで分割して管理している．サーバやスイッチの故障に備えて，各チャンクは異なるスイッチ配下にある複数のサーバ上に複製が保存される．基本的な複製作成のタイミングはデータ書き込み時で，書き込み対象データを保存する複数のサーバに転送し，データが複数のサーバに行き渡ったことを確認した後に，実際の書き込みを指示する方式となっている．

分散テーブルシステムでは，WAL(ログ先行書き込み) 方式を採用しており，分散テーブルへの書き込み要求を受信すると，分散ファイルシステム上にあるコミットログに対して追記を行う．追記が完了するとメモリ上のデータを更新し，書き込み要求元へに応答を返す方式になっている．

前述の CBoC タイプ 2 を用いた性能試験を実施したところ，分散テーブルの書き込み性能が低いという現象に見舞われた．小さな Value の書き込みは想定通りの性能で書き込めるのだが，大きな Value の書き込みは想定通りの性能が出ず，データ量で換算しても，小さな Value 書き込み時のスループットに追いつけない状態だった．

書き込む Value を大きくすると書き込み性能が劣化する現象について，アーキテクチャやリソースの限界によるものなのか，それとも，CBoC タイプ 2 のサブシステム内またはライブラリや OS に問題があるのか特定し，その性能が妥当かどうか確認するため，本事象を問題とした．

3. 問題の分析

分散テーブルへの書き込み性能の劣化という問題に対して，本節では分析を行い，原因箇所を切り分けていくことで，Linux カーネル内のセマフォの競合に原因があることを突き止める．

以降では，性能劣化の原因を特定するまでの分析手順を示す．

3.1 ログ分析による原因の切り分け

最初に，原因が分散テーブルシステム自体にあるのか，それとも分散ファイルシステムなどの別な CBoC タイプ 2 の構成要素にあるのかを切り分けるために，分散テーブルシステムの詳細ログを分析した．

その結果，分散ファイルシステム上にあるコミットログの書き込み性能が原因であることを特定した．

分散テーブルシステムでは，WAL 方式を採用しているため，分散テーブルへの書き込み性能は，分散ファイルシステムへのコミットログ書き込み応答時間に大きく左右される．そのため，分散ファイルシステムへの書き込み性能の劣化が，分散テーブルへの書き込み性能の劣化という事象になってあらわれた．

3.2 分散ファイルシステム内の原因の特定

分散テーブルにおいて書き込み性能が出ない原因は，コミットログの書き込みに時間がかかっているためであった．コミットログは分散ファイルシステム上に存在するため，以下のような仮説をたてた．

- ハードディスクへの書き込みが行われるので，読み書きの多重度や別なアプリケーションの利用状況によって応答時間が変動する
- ツリー型のネットワークトポロジであったため，エッジスイッチ・コアスイッチ間で輻輳している
- データの複製を保持する複数のサーバの何れかが，高負荷状態に陥る
- 書き込み要求が分散されず，特定のサーバに集中する

分散ファイルシステムへの書き込みは，ハードディスクやネットワークといった他の処理による影響を受けやすい I/O 処理を多く含むほか，複製作成のために複数のサーバ間で同期をとりながら処理を行うため，複数のサーバのうち 1 台でも応答時間が悪化すると，書き込み性能が容易に劣化する．そのため，上記の仮説を立て，その仮説が正しいかどうか検証することにした．

まず，本問題は再現性が高く，問題が発生するソフトウェアのソースコードに直接手を加えられる環境であったため，書き込み性能に影響があるプロトコルの処理部分に処理時間計測のコードを埋め込み実行することで，時間のかかっている処理の特定を試みた．

埋め込む基準は以下の通りで，上述の予想が正しいことが検証できる部分のほか，いくつか性能低下の原因となりそうな箇所を抽出した．

- ファイルの読み書き箇所:
  open, close, mmap, munmap, memcpy などのシステムコール・関数．CBoC タイプ 2 の分散ファイルシステムでは，ファイルをメモリにマッピングして読み書きを行うため，memcpy 等も対象
- RPC 応答時間

表 1 プロトコルの処理時間内訳 (抜粋)

リクエスト受信	4ms
malloc	288ms
RPC 応答時間	616ms
レスポンス送信	13ms

- ソケットの読み書き時間
- 赤黒木などのデータ構造操作箇所:
  扱うデータ量の増加に伴って，効率の良いデータ構造でも性能が低下する可能性を考慮
- ロック獲得待ち時間:
  マルチスレッドアプリケーションであるため，Mutex などのロック競合が起きている可能性を考慮
- メモリ操作箇所:
  malloc, memcpy, memset システムコール・関数．大きなデータのメモリコピーが起こる可能性を考慮

その結果，プロトコルの応答時間のうち，ファイルの読み書き箇所や，RPC 応答時間などは予想通り大きな割合を占めていたが，malloc などのメモリ操作箇所も，大きな割合を占めていた．表 1 に，あるプロトコルの処理時間の内訳を示す．

RPC 応答時間の中にはソケットの読み書き時間と，リモートサーバでの処理時間も含むため，長くなることは想定できるが，malloc に 300ms 近くかかり，処理時間の 3 割近くを占めているのは予想外であった．最悪の場合は，malloc に十数秒要している場合もあった．

リモートサーバでの処理の中に malloc が含まれる場合，RPC 応答時間も malloc 応答時間の影響を受ける．そのため，本問題の原因は分散ファイルシステムではなく，メモリ確保に時間がかかっているのが原因であると考えた．

3.3 malloc に時間がかかる原因の分析

malloc 関数は，Linux では一般的に glibc の実装が利用される．マルチスレッドにおける glibc malloc はスケーラビリティが低いことが指摘されており，tcmalloc[6] や jemalloc[7] などの代替となるような実装が提案されている．そのため，malloc に時間がかかる原因を分析するにあたり，原因が glibc にあるのか，Linux カーネルにあるのかを切り分けることとした．

分散ファイルシステムのうち，本問題に関係するプロセスは，外部からの要求を受けて，ファイルを開き，メモリにマッピングし，先読み指示を発行し，ファイルの内容を送信／受信したデータを書き込む処理を主として行っている．

該当プロセスのサブセットとなる，上記処理のみを実施するテストプログラムを作成し，負荷を加えたところ，同様に malloc で時間がかかる現象が見られた．

そのため，テストプログラム実行中にマジック SysRq を利用して実行中タスクのバックトレースを取得することで，原因が glibc malloc にあるのか Linux カーネルにあるのかを分析した．

```
Call Trace:
 [<ffffffff800336ac>] submit_bio+0xe4/0xeb
 (中略)
 [<ffffffff80067b55>] do_page_fault+0x4cb/0x874
 [<ffffffff80063ff8>] thread_return+0x62/0xfe
 [<ffffffff8005ede9>] error_exit+0x0/0x84
(中略)
Call Trace:
 [<ffffffff800656ac>] __down_read+0x7a/0x92
 [<ffffffff80067ad0>] do_page_fault+0x446/0x874
 [<ffffffff80063ff8>] thread_return+0x62/0xfe
 [<ffffffff8005ede9>] error_exit+0x0/0x84
(中略)
Call Trace:
 [<ffffffff8005aa74>] hrtimer_cancel+0xc/0x16
 [<ffffffff80064d05>] do_nanosleep+0x47/0x70
 [<ffffffff80065613>] __down_write_nested+0x7a/0x92
 [<ffffffff800239c7>] sys_mmap+0x6d/0xc1
 [<ffffffff8005e28d>] tracesys+0xd5/0xe0
```

図 2 マジック SysRq によるバックトレース取得結果

その結果，図 2 の様に，複数のスレッドがセマフォの獲得待ちで待機していることがわかった．そして，Linux カーネルのソースコードより，競合しているセマフォは，mmap_sem という名前のセマフォであることを突き止めた．

4. 問題の原因

4.1 Linux におけるプロセスアドレス空間

Linux におけるプロセスのアドレス空間は，他のプロセスとは独立したリニアアドレス空間になっており，カーネルはメモリリージョンという単位でリニアアドレス空間より実際に利用する区間を管理する．

メモリリージョンの生成や削除には, execve や fork, _exit といったプロセスの生成・削除に伴うシステムコールや，brk や mmap，munmap などヒープの拡張やファイルをメモリにマッピングするシステムコールを利用することによって行われる．

メモリリージョンは片方向リンクリストと赤黒木によって管理され，これらの構造体の排他制御の為にセマフォが用いられる．このセマフォはプロセスアドレス空間を管理する mm_struct 構造体の mmap_sem という名前のメンバである．このセマフォはメモリリージョンの追加や削除が行われるときは排他ロックされ，メモリリージョン内の読み書きを行う場合は，操作中のメモリリージョンが，他の処理によって削除されたりすることが無いように共有ロックされる．

4.2 mmap_sem セマフォの競合

今回の問題は mmap_sem セマフォのロック獲得が競合

© 2012 Information Processing Society of Japan

したために発生した．mmap_sem はページフォルトを伴うメモリ空間へのアクセスが発生した場合に共有ロックを獲得する．また，mmap を用いてファイルをメモリ空間にマッピングした場合，該当メモリ空間へ読み込みアクセスを行うと，ページフォルトハンドラ内でファイルシステムからデータを読み込み，ページキャッシュ内のページを該当メモリ空間に割り当てるという作業を行う．そのため，ファイルシステムからデータを読み終えるまで mmap_sem セマフォの共有ロックを保持し続けることになり，その間の排他ロック獲得処理はブロックされる．

mmap_sem セマフォの排他ロックを必要とする処理は，メモリリージョンの生成や削除などである．他のファイルを mmap システムコールを利用してメモリに割り当てようとする処理などが該当するが，glibc の malloc 内部でも brk や mmap といったシステムコールが利用されている．そのため，malloc を利用した場合でも mmap が呼び出される場合がある．

以上より，ファイルをマッピングしたメモリ空間でのページフォルトが多発した場合，後続の mmap 等は mmap_sem セマフォの排他ロック獲得待ちで待たされ，排他ロック獲得待ちがある場合は，後続の共有ロックも待ち状態に入る．そのため，mmap 等の mmap_sem セマフォの排他ロックを必要とする処理よりも前に発生した全てのページフォルトハンドラの処理が終わらない限り，mmap 等の処理は待たされ，mmap 等の処理よりも後続のページフォルトハンドラも待たされることになる．

4.3 madvise における mmap_sem

ファイル読み込みのスループットを向上させるために，madvise システムコールに MADV_WILLNEED を指定して呼び出すことにより，カーネルに先読みをアドバイスすることができる．madvise では mmap_sem セマフォの共有ロックを獲得してから，指定されたメモリ範囲でページキャッシュに無いデータをファイルシステムより先読みする．

madvise を用いた先読みでは，現在の Linux カーネル実装では通常の先読みと異なった動作をする．通常の先読みでは，バッキングストアデバイス (HDD など) の読み込み要求キューが一杯である場合，先読みをキャンセルすることで，フォアグラウンド処理に影響を与えないような動作になっている．しかし，madvise を用いた先読みでは，読み込み要求キューが一杯であっても，先読みを強制的に実施する．そのため，要求キューが一杯でない場合はキューに積む処理をして応答が戻るため，短い時間で madvise の処理が完了するのに対し，要求キューが一杯の場合は，要求キューに空きができるまで mmap_sem セマフォの共有ロックを確保したまま madvise の処理は止まってしまう．

madvise は上記のような動作となっているため，低い負荷の時など，バッキングストアデバイスの読み込み要求キューに空きがある場合は，は madvise はすぐに完了し，mmap_sem セマフォの共有ロックを長時間保持しない．しかし，読み込みスレッドが多い場合や，他のプロセスでファイルの読み込みを行っている場合など，負荷が高いときはバッキングストアデバイスの読み込み要求キューが埋まっていることが考えられる．その場合，madvise 処理は読み込み要求キューが空くまでブロックするため，mmap_sem セマフォの共有ロックを長時間保持し続けることになる．

5. 影響範囲

この問題は，ファイルがマップされたメモリ領域を読み書きするスレッドと，明示的 (mmap 等)・暗黙的 (malloc 等) に関わらず，メモリリージョン操作を行うシステムコールを発行するスレッドが競合する場合に起こりうる．

多くの場合，スレッド内の処理にて malloc を利用しているため，マルチスレッドかつ，ファイルをメモリにマッピングして読み書きするアプリケーションが影響を受けると考えられる．特に glibc malloc の場合，おおよそ 128KB 以上といった比較的大きいサイズのメモリ領域を確保しようとすると確実に mmap(2) を呼び出してしまうため，影響を受けることが多くなる．

これらの問題のうち，ページフォルト中にファイルシステムへのアクセスが生じる場合に関しては，Linux カーネル 2.6.37 にて改善がなされている．カーネル 2.6.37 以降では，ページフォルト中にファイルシステムへのアクセスが生じる場合は，mmap_sem セマフォの共有ロックを一度手放して，ページキャッシュにファイルのデータを載せ，再度，mmap_sem セマフォの共有ロックを確保して，ページの割り当てを行っている．しかし，MADV_WILLNEED を指定して madvise システムコールを呼び出す場合は，現時点での最新カーネル (バージョン 3.3.0) でも同様の動作となる．

そのため，本問題の影響範囲は，先述した動作を行うアプリケーションかつ，Linux カーネル 2.6.37 よりも古いカーネルを利用している場合が該当する．なお，代表的な Linux ディストリビューションである Red Hat Enterprise Linux(RHEL) では RHEL6.1 の kernel-2.6.32-131 以降では，パッチがバックポートされているが，RHEL5 系では，パッチのバックポートは行われていない．RHEL5 系は 2017 年 3 月 31 日までサポートが行われるため，当面は RHEL5 系を利用するユーザも多いものと思われる．

本問題と影響するカーネルバージョンを表 2 にまとめた．本稿で提案する回避策は，表 2 中の未対処部分に対応するものである．

5.1 分散ファイルシステム・分散テーブルへの影響

我々が開発した分散ファイルシステムでは，ユーザ空間

表 2 Linux カーネルバージョンと本問題の対処状況

	2.6.37 未満	2.6.37 以降
madvise(MADV_WILLNEED) 利用時の長時間ロック	×	×
ページフォルト時の長時間ロック	×	○ (対処済)

へのメモリコピー削減による性能向上や，ファイルをメモリと同様に扱える利便性を重視して，mmap を使いファイルをメモリ空間にマッピングしていた．そして，読み込み時のスループット向上を目的としてチャンクサイズ単位で MADV_WILLNEED を指定した madvise を呼び出していた．

また，分散ファイルシステムへの書き込み要求があった場合は，複製を保持するサーバのメモリ上に一度転送し，データが行き渡ったことを確認してからディスクへ書き込む指示を書きこみ要求元が発行することで，全複製間での一貫性を保つ方式となっているため，比較的大きなメモリ確保を glibc malloc に要求していた．

我々の分散ファイルシステムでは，上述した mmap_sem セマフォの競合を起こしやすいシステムコール・ライブラリの使い方を利用していたため，性能の劣化が著しかった．

このことは，第 2 節で述べた本問題の事象に合致する．分散テーブルへ大きなデータを書き込むと，コミットログに書き込むデータサイズも大きくなり，分散ファイルが glibc malloc に要求するメモリサイズも大きくなる．その結果，malloc 内で暗黙的に mmap が呼び出される頻度が高くなり，mmap_sem セマフォとの競合が発生しやすい状況に陥ったと考えられる．

6. 回避策

本問題は Linux カーネルの問題であるため，madvise を利用しない場合は最新のカーネルに更新することで問題が改善する．しかし，以下の場合など容易にカーネルを入れ替えられない場合が考えられる．

- 上位で動作させるアプリケーションとの制約により，Linux カーネルやディストリビューションの更新が実施できない
- IaaS などのクラウド環境を利用しているためディストリビューションや Linux カーネルの選択に制限がある

そのため，本問題をアプリケーション側で回避する方法について検討を行った．

本問題は，ページフォルト時に mmap_sem の共有ロックを確保したまま，排他ロックを必要とするシステムコールを発行することによって起こる．よって，回避策の方針としては以下の二つが考えられる．

(1) ページフォルト中にファイルシステムへのアクセスなど長い時間，共有ロックを保持するような処理が行われないようにする
(2) 排他ロックを必要とするシステムコールの発行を最小限に止める

以上の方針に基づくと具体的な回避策は以下のようになる．実際の開発では，改修に要する工数も重要な要素なので，それぞれの回避策について，想定される改修規模も付記した．

(回避策 1) MADV_WILLNEED を指定して madvise を呼び出さない (改修規模:小)
(回避策 2) メモリプールをアプリケーション側で管理し，mmap を呼び出すような malloc の呼び出しを削減する (改修規模:中)
(回避策 3) ファイルをメモリにマッピングして読み込むのではなく，read システムコールや fread を利用する (改修規模:大)

(回避策 1) によって，一度の mmap_sem の共有ロック内で，大きなデータを読み込むことにより，排他ロックの確保が長時間待たされるのを防ぎ，待ち時間を分散させることで，応答時間の短縮を見込む．madvise を呼び出している箇所を削除するだけで改修が可能であるため，改修規模はもっとも少ない．

(回避策 2) では，mmap を呼び出すような malloc は，比較的大きなメモリ領域を確保しようとする場合に多く発生するので，アプリケーションの中で malloc を呼び出している箇所のうち，大きなメモリ領域を確保する可能性のある箇所のみをメモリプールから確保するように修正する．大きなメモリ領域の確保を行う箇所は，それほど多くないと考えられるので，中程度の改修規模で排他ロックの回数を削減することを見込む．

(回避策 3) では，ページフォルトハンドラ内でファイルシステムへのアクセスが発生しなくなるが，メモリ上マップされたファイルを読み書きしていた箇所を read/write などのシステムコールに書き換えなければならない．また，バッファの管理なども必要になる．そのためもっとも改修規模が大きな回避策となる．

また，回避策 1 と 2，回避策 2 と 3 は組み合わせることができるので，改修規模と効果のバランスでどの回避策をとるべきかを選択する．

次節でこれらの回避策の効果について評価を行う．

7. 回避策の評価

7.1 評価方法

それぞれの回避策によって，どの程度の効果があるのか，ベンチマークを実施した．ベンチマークの内容は，ファイル読み込む複数のスレッドと，mmap/munmap を発行するスレッドを同時に実行し，mmap システムコールの応答時間にどのような変化があるかを測定した．

評価を行ったファイル読み込み方法と回避策の組み合わせは以下のとおり．

表 3 評価環境

CPU	Intel Core i7 870
メモリ	DDR3-SDRAM PC3-10600 16GB
HDD	Samsung HD103SJ
ファイルシステム	ext4
カーネル	2.6.18-308.el5

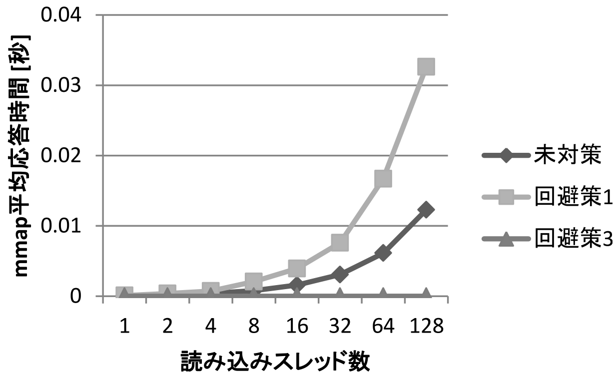

図 3 要求サイズ 64KB 時の平均応答時間

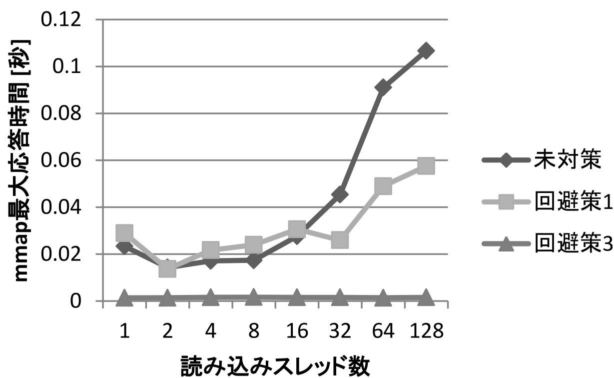

図 4 要求サイズ 64KB 時の最大応答時間

(未対策) mmap を利用した読み込み (madvise による先読み指示あり)

(回避策 1) mmap を利用した読み込み (madvise による先読み指示無し)

(回避策 3) read を利用した読み込み

回避策 2 については，本ベンチマークを用いた場合，回避策 1，3 と直接比較することができないため，我々が開発したシステムにおける効果のみを測定し，第 8 節にて別途評価を行う．

7.2 評価環境

評価には，RHEL の互換ディストリビューションである CentOS の 5.8 を用いて実施した．詳細は表 3 に示す．

7.3 評価結果

Linux カーネル 2.6.18-308 での評価結果を図 3 から図 6 に示す．これらの図の縦軸は，mmap/munmap 発行スレッドにおける mmap の応答時間を示しており，横軸は読み込みスレッド数となっている．

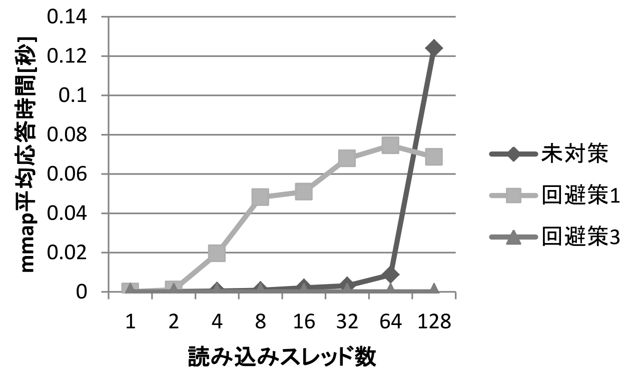

図 5 要求サイズ 16MB 時の平均応答時間

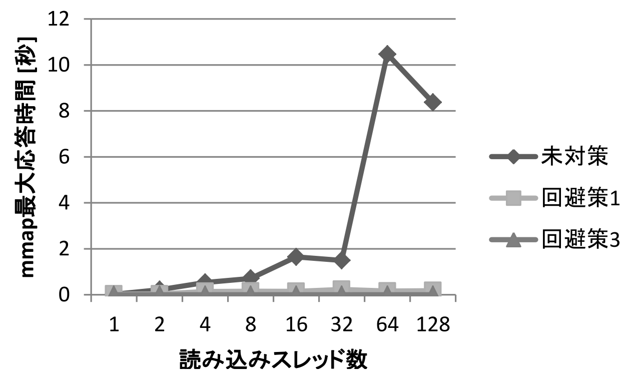

図 6 要求サイズ 16MB 時の最大応答時間

図 3 から図 6 より，ページフォルト中に mmap_sem セマフォの共有ロックを確保したまま，ファイル読み込みを行う (未対策) や (回避策 1) は，mmap_sem セマフォの排他ロックを必要とする mmap システムコールの応答時間が悪化することが分かる．読み込みスレッドが複数ある場合は，共有ロックを獲得した複数のスレッドが同時にファイルの読み込みを行うため，排他ロック獲得スレッドは，排他ロック獲得要求を出す前に要求された全てのファイル読み込み処理の完了を待たなければ，ロックを獲得できないため，長い時間待機しなければならない．そのため，読み込みスレッド数が増加すればするほど，mmap の応答時間は悪化していく．

図 3 や図 5 では，回避策 1 を適用するよりも，未対策の方が平均応答時間が短くなっている．しかし，図 4 や図 6 では，その傾向が逆転しており，回避策 1 の方が最大応答時間は短い．これは，madvise によって一度に多くのデータを読み込むことによって，mmap_sem セマフォの衝突回数が減少し平均応答時間は短くなるが，セマフォが競合すると，madvise では多くのデータ読み込みのために，長い時間ロックを獲得し続けるためである．また，読み込みスレッド数が多くならない限りは，バッキングストアデバイスの要求キューに空きがある状態であるため，読み込みスレッド数が少ない場合は，未対策と回避策 1 はほぼ同じ応答時間となる．

ファイルをメモリにマッピングしてファイルを読み込む

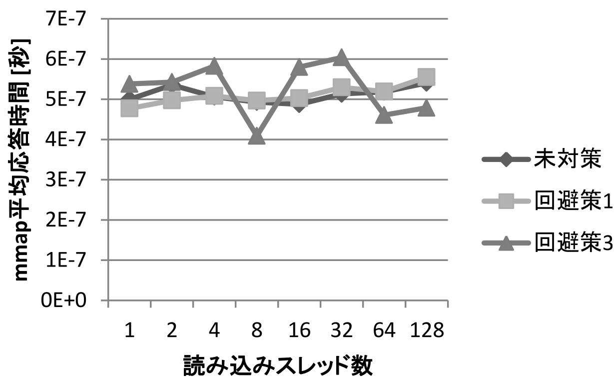

図 7 要求サイズ 64KB 時の平均応答時間 (Kernel 3.3.0)

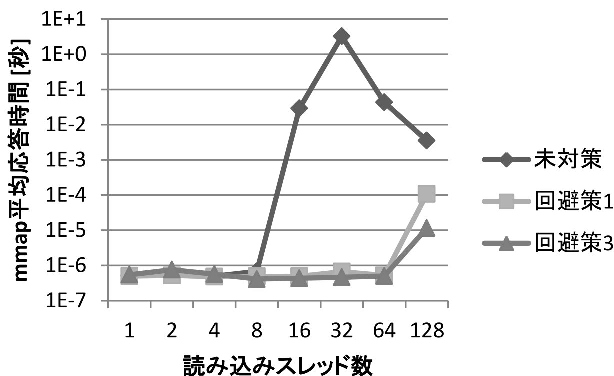

図 8 要求サイズ 16MB 時の平均応答時間 (Kernel 3.3.0)

方式に対して，read システムコールを用いたファイル読み込み方式では，mmap_sem セマフォに関与しないため，mmap などのメモリリージョン操作を行うシステムコールの性能に影響を及ぼさない．そのため，図 3 から図 6 にて示されるように，(回避策 3) は，読み込みスレッド数や，システムコールに渡すデータサイズが変化しても，mmap 応答時間に影響を与えない．そのため，read システムコールを利用する場合は，ファイルをメモリにマッピングして読み込む場合と比べておおよそ 100 分の 1 の応答時間で安定して mmap を実行できている．

7.4 Linux カーネル 3.3.0 での評価結果

バージョン 2.6.37 以降のカーネルでは，ページフォルト時にファイルからの読み込みを行う場合は，mmap_sem セマフォの共有ロックを一度解放するように改善された．

改善されたカーネルのうち, 現時点で最新版である Linux カーネル 3.3.0 を用いて，同様の評価を行った．評価結果を図 7 から図 9 に示す．

新しいカーネルでは，madvise を利用しない読み込み方法と，read システムコールを利用した読み込み方法とで，mmap の応答時間はほぼ同じになっており，2.6.18-308.el5 と比較すると大幅に改善されている．しかし，新しいカーネルにおいても madvise の動作は変わっていないため，図 8 や図 9 にて示されるように，読み込みスレッド数が多い場合に mmap の応答時間は悪化している．今回の評価では

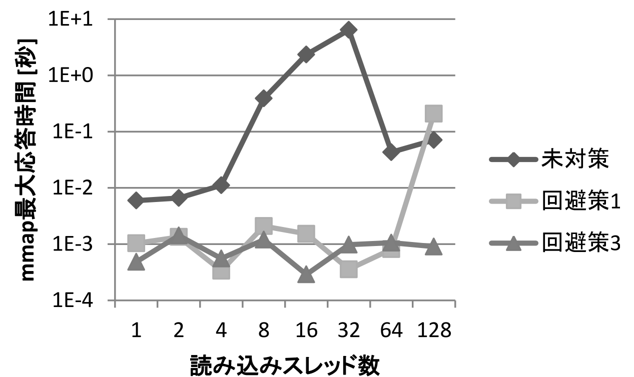

図 9 要求サイズ 16MB 時の最大応答時間 (Kernel 3.3.0)

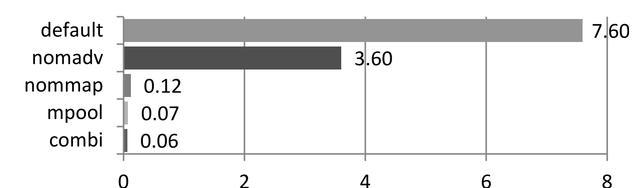

図 10 分散ファイルシステムにおける malloc 応答時間 (平均)[ms]

要求サイズが 64KB の場合は未対策の場合でも平均・最大応答時間が劣化することは確認出来なかったが，madvise の動作は変わっていないため，バッキングストアの要求キューの状態に依っては，要求サイズが小さい場合においても，古いカーネルと同様の現象が見られると思われる．

8. 大規模分散処理基盤への回避策適用による効果

第 7.3 節では，ベンチマークによって，各回避策の効果について評価を行った．本節では，実際のソフトウェアに回避策を組み込んだ時にどのように性能が向上するか評価を行う．また，ベンチマークではメモリプールの効果を他の回避策と一律に比較できなかったため，評価対象外としていたが，本評価ではメモリプールを含めた全ての回避策について評価を行った．

本評価では分散ファイルシステムに 4 種類の回避策の組み合わせを適用した．そして，分散ファイルシステム内での malloc 応答時間や，分散ファイルシステムに対して書き込みを行う分散テーブルシステムのコミットログ書き込み処理の応答時間を計測した．結果を図 10 から図 15 に示す．各系列の説明は表 4 に記述した．

なお，本評価は RHEL5(kernel-2.6.18-194.el5) で実施し，サーバ 3 台という小規模な環境で実施した．分散テーブルへデータを書き込むツールとして，KeyValue ストアを対象とした代表的なベンチマークツールである YCSB[8] を利用した．ハードウェアは表 3 とは異なる環境で行ったが，詳細については割愛する．

図 10 から図 12 の様に，古いカーネルであっても，madvise を無効化することによって malloc 応答時間に改善がみられる．combi においては，図 12 を見ると，default の

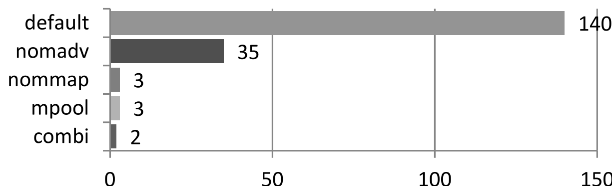

図 11 分散ファイルシステムにおける malloc 応答時間 (99%ile)[ms]

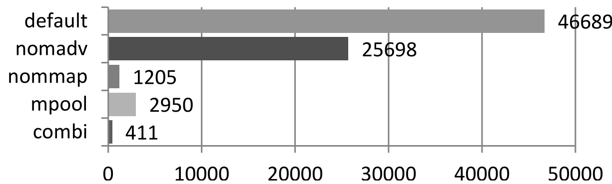

図 12 分散ファイルシステムにおける malloc 応答時間 (最大)[ms]

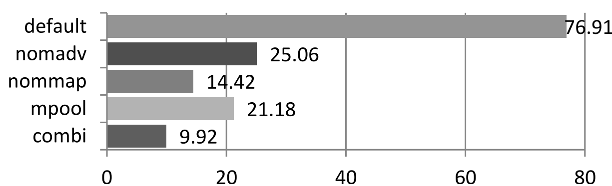

図 13 分散テーブルシステムにおけるコミットログ書き込み応答時間 (平均)[ms]

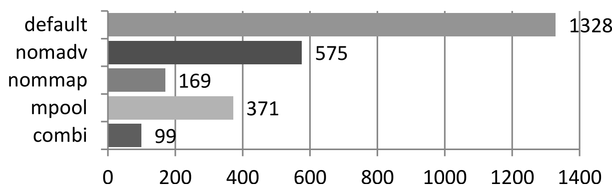

図 14 分散テーブルシステムにおけるコミットログ書き込み応答時間 (99%ile)[ms]

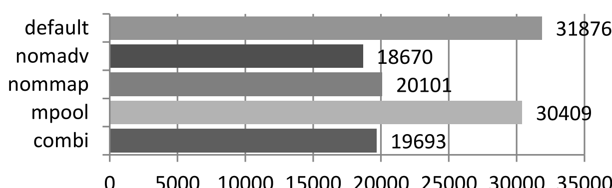

図 15 分散テーブルシステムにおけるコミットログ書き込み応答時間 (最大)[ms]

表 4 系列の説明

default	回避策を組み込まない
nomadv	回避策 1
nommap	回避策 3
mpool	回避策 1 + 2
combi	回避策 2 + 3

表 5 回避策と改修規模，効果のまとめ

改修案	想定される改修規模	malloc 平均応答時間	実 AP 平均応答時間	実 AP99%ile 応答時間
回避策 1	小	50%短縮	67%短縮	57%短縮
回避策 1+2	中	99%短縮	72%短縮	72%短縮
回避策 3	大	98%短縮	81%短縮	87%短縮
回避策 2+3	大	99%短縮	87%短縮	92%短縮

100 分の 1 にまで短縮している．

また，分散ファイルシステムにおける malloc 応答時間が改善したため，図 13 から図 15 で示されるように，分散テーブルシステムにおける，コミットログ書き込み応答時間が改善した．

9. 改修規模と効果

第 8 節で行った評価と，各回避策の改修規模をまとめると，表 5 のようになる．

改修規模が大きい物ほど効果は大きいが，実際のソフトウェア開発では，効果の差が改修規模に見合うかどうかを考慮に入れて検討する必要がある．

10. おわりに

本稿では，アプリケーションの性能劣化事象について分析し，Linux のメモリ管理に起因したセマフォの競合に原因があることを突き止め，報告した．本事象は最新の Linux カーネルを用いても一部改善されないケースがあること，および，カーネルの変更ができないようなシステムが存在することを想定し，アプリケーションの修正による回避策について検討・評価を行った．その結果，検討した回避策はいずれも効果があり，また改修規模に比例した効果が得られることが明らかになった．これにより，本問題を抱えるアプリケーションは，アプリケーションの特性に合わせて，改修規模と効果のバランスのとれた回避策を採用することが可能となる．

今後は，madvise の問題を解決するパッチの検討を行ったり，他にも Linux に起因するバグや性能問題に遭遇していないか，引き続き分析し，結果をコミュニティにフィードバックできないか検討する予定である．

参考文献

[1] 鷲坂, 中村, 高倉, 吉田, 冨田: 大量データ分析のための大規模分散処理基盤の開発, NTT 技術ジャーナル, Vol.23, No.10, pp.22–25, (2011).

[2] Ghemawat, Sanjay. Gobioff, Howard. and Leung, Shun-Tak.: The Google File System, Proc. of the 9th ACM Symposium on Operating Systems Principles, ACM, pp.29–43, (2003).

[3] The Apache Software Foundation: Hadoop Distributed File System, http://hadoop.apache.org/hdfs/.

[4] Chang, Fay. Dean, Jeffrey. Ghemawat, Sanjay. Hsieh, Wilson C. Wallach, Deborah A. Burrows, Mike. Chandra, Tushar. Fikes, Andrew and Gruber, Robert E.: Bigtable: a Distributed Storage System for Structured Data, Proc. of the 7th USENIX Symposium on Operating Systems Design and Implementation, USENIX Association, (2006).
[5] The Apache Software Foundation:
Apache HBase Project, http://hbase.apache.org/.
[6] S. Ghemawat and P. Menage.: Tcmalloc: Thread-caching malloc,
http://gperftools.googlecode.com
/svn/trunk/doc/tcmalloc.html.
[7] Jason Evans.: jemalloc,
http://www.canonware.com/jemalloc/.
[8] Cooper, Brian F. Silberstein, Adam. Tam, Erwin. Ramakrishnan, Raghu. and Sears, Russell.: Benchmarking cloud serving systems with YCSB, Proceedings of the 1st ACM symposium on Cloud computing, ACM, pp.143–154, (2010).

© 2012 Information Processing Society of Japan