Vol.2015-OS-135 No.10
Vol.2015-EMB-39 No.10
2015/11/24

パフォーマンス障害に対する管理図によるログ削減手法

中村 啓太郎[1]　深井 賢[1]　吉村 剛[1]　河野 健二[1]

概要：本稿は，Linux において記録されるログの削減手法を提案したものである．近年，あらゆるコンピュータシステムに対して，高い信頼性が求められるようになっている．システムの信頼性が損なわれる要因のひとつにパフォーマンス障害がある．パフォーマンス障害とは，性能設計上想定していた場合に比べシステムの応答時間が遅くなり，処理効率が悪くなることを指す．こうしたパフォーマンス障害の原因特定には，一般的にログを分析する方法が用いられている．しかし，すべてのイベントをログとして記録した場合，CPU 利用率が増加し，記録されるログサイズが著しく増加するという問題がある．本稿では，管理図により性能異常と判定されたログのみを取得することによるログ削減手法を提案する．国内の IT ベンダにおける社内システムにおける障害事例に適用したところ，ログサイズを約 1/10 に削減することができた．

1. はじめに

様々な情報サービスを一般のユーザが利用するようになり，いわゆるミッション・クリティカルなシステムに限らず，あらゆるコンピュータシステムに対して高い信頼性が求められるようになっている．しかし，高い信頼性を達成することは決して容易ではない．信頼性が損なわれる要因のひとつとしてパフォーマンス障害というものがある．

パフォーマンス障害とは，システムの応答時間が遅くなったり，スループットが低下したりして，期待した性能が達成できない状況を指す．そうしたパフォーマンス障害は，システムを利用するユーザに深刻な影響を与えることが知られている．2012 年 10 月には， Amazon EC2 で 7 時間ほどパフォーマンス障害が発生し，その間 Reddit などの有名サイトがダウン，多くのサービスが停止に追い込まれている [1].

パフォーマンス障害は再現性が低く，原因特定が難しいことが多いという側面を持つ．これはパフォーマンス低下の原因要素が多岐にわたる上，クラッシュなどと異なり障害発生の瞬間が把握しづらいためである．

詳細な動作履歴がログに残っていれば，それを手がかりに原因を究明することが可能になる．しかし，十分に詳細なログを取得するためには，CPU 利用率，ログサイズの増大が発生する [2]．そこで，ログの取得の頻度やサイズを小さく抑えることが必要となる．

本稿では，すでに文献 [2] で提案したパフォーマンス障害に対するログ削減手法を Linux 2.6.34.7 に実装した．パフォーマンス障害の要因分析に有用であると思われるログのみを選択的に取得・記録する．パフォーマンス障害の中でも特に，ディスク I/O に関する障害を対象とし，なかでもキューの滞留時間の遅延を監視する．パフォーマンス障害が発生する場合，処理が異常に遅い部分が原因である可能性が高い．そこの処理を選択的に記録することで，詳細な記録を行う場合に比べ，CPU 利用率を抑え，ログサイズを削減することができる．処理が遅い部分を検知するために，統計学的に確立された異常検出手法である管理図 [3] という手法を用いる．これを利用し，パフォーマンス障害を検知し，それに関するログを取得することで，ログを削減する．

具体的には，Linux カーネルに標準で搭載されるトレースツールである ftrace [4] のコードに手を加えることで，実装した．書き込み部分にフックをしかけ，管理図を組み込んだ．

提案手法の有効性を示すため，国内の IT ベンダにおける社内システムにおける障害事例を用いて実験を行なった．管理図を用いることで，CPU 利用率を抑えることができ，ログサイズを 約 1/10 に削減できた．

本稿の構成は以下の通りである．2 章では，ロギングによるオーバーヘッドについて説明する．3 章では，管理図について説明する．4 章では，管理図によるログ削減について説明する．5 章で評価実験を行い，6 章で関連研究について紹介し，7 章で結論を述べる．

[1] 慶應義塾大学大学院
Keio University Graduate School

2. ロギングによるオーバーヘッド

パフォーマンス障害が発生した場合，開発者は直ちに解析を行う．その方法としてログを用いた解析手法がある．システムの挙動をログに記録しておき，異常の原因を調査するという方法である．しかし，この方法にはすでに文献 [2] で報告したとおり，CPU 利用率の増大，生成されるログのサイズが大きいという問題が発生する．

図 1 にログを取得した場合と取得していない場合の CPU 時間の差を示す．横軸がベンチマークの実行時間，縦軸が CPU 時間である．ftrace によりログを取得した場合，全計測時間を通して CPU 時間に関して約 20 %のオーバーヘッドがあった．ログ取得によるオーバーヘッドは 5% 以下が望ましいという要望があり，ログをすべて取得したのではその要望を満たすことができないことがわかる．

図 2 に記録されたログのサイズを示す．横軸がベンチマークの実行時間，縦軸がログサイズである．1 秒で 2GB，20 秒で 6GB を超えるログが記録されていた．

上記の問題により，すべてのログを記録するのではなく，原因究明に必要なログだけを選択的に記録する必要がある．

3. 管理図

本稿では，管理図 [3] という統計的手法を利用して，パフォーマンス障害を検知する．管理図とは，統計学的に確立された異常検出手法である．元々，製品製造工程や経営工程を管理するためのものであり，工程が統計的に正常か異常かを判定するものである．過去の管理対象の値（本稿では I/O 処理時間）の分布と，現在の管理対象の値の分布を統計的に比較し，二つの分布が同じか異なるかを判定する．詳しくは文献 [2] を参照されたい．

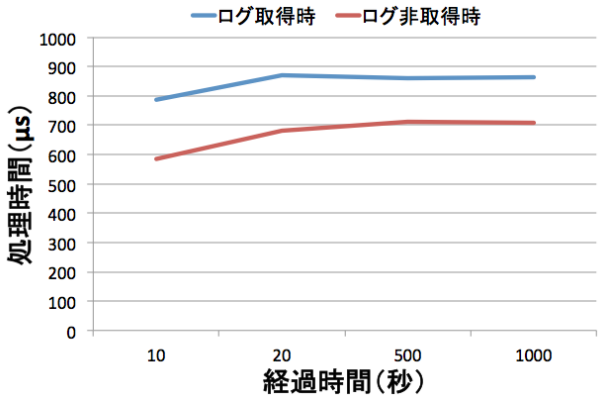

図 1 文献 [2] より再掲：ロギングによるオーバーヘッド

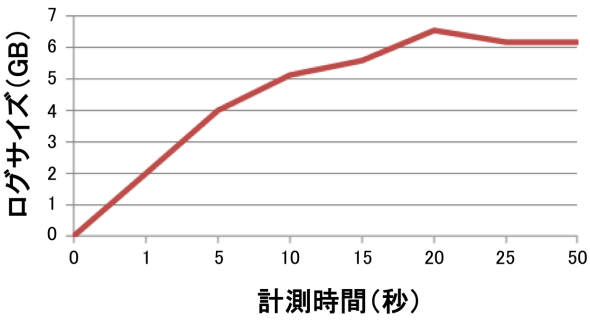

図 2 文献 [2] より再掲：ログサイズ

ここでは，図を用いて簡単に説明しておく．図 3 に管理図の例を示した．横軸が経過時間，縦軸が監視対象の処理時間である．このように監視対象の処理時間をプロットしていく．その際，工程が正常か異常かを判定するために，図 3 にある通りあらかじめ過去に測定したデータから中心線，基準線を引いておき，基準線を超えた場合に異常と判定する..

また，管理図は監視対象の処理時間が基準線を超えない場合でも，何らかの偏りや傾向がある場合には，異常が発生している可能性があると判定する．図 4 に管理図の判定パターンを示す．左側の緑枠で囲った部分は正常と判定する場合である．異常と判定するパターンとして，例えば，真ん中の赤枠で囲った部分のように監視対象の処理時間が中心線と基準線の間の値ばかり取る場合，また右側の赤枠で囲った部分のように監視対象の処理時間が単調増加している場合がある（これらの詳しい内容については，参考文献 [5][6] を参照）．

4. 管理図によるログの削減

4.1 概要

管理図という統計的手法により異常であると判定されたログのみを取得することで，ログを削減する．システムの通常稼働時のパフォーマンスから管理図を作成し，システムの挙動を監視する．処理時間が通常より長いものを異常と判定し，その部分のログを選択的に取得する．また，障害発生時のみのログだけでなく，障害発生前のログも取得する．

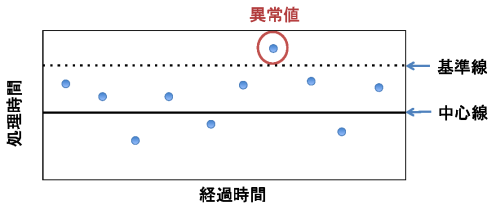

図 3 管理図の例

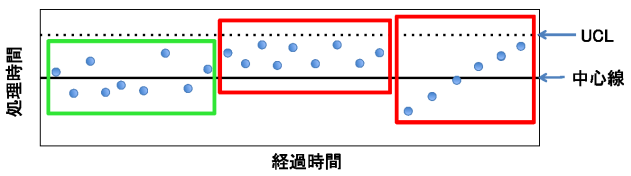

図 4 管理図の判定パターン

4.2 障害発生前のログ

管理図によって異常と判定された値と，その少し前の値をログに残すようにすることで，ログの削減を行なう．文献 [2] で説明した手順により作成した管理図を用いて，システムの挙動を監視する．そして，UCL を超えた部分のログと，その少し前のログを残す．

管理図によって異常を検知した場合，異常が発生した瞬間のログのみを記録したのでは原因究明には不十分なことが多い．なぜなら，異常が発生した瞬間のログ以外を記録しなかった場合，何らかの根本原因 (root cause) から異常が発生するまでの期間のログが残らず，異常発生の原因をさかのぼって調べることができなくなるためである．

例えば，図 5 のようなパフォーマンス障害が発生したとする．横軸が経過時間，縦軸が I/O 処理時間である．図を見ると明らかなように，40 分経過時の I/O 処理時間が異常に遅くなっており，パフォーマンス障害が発生していることがわかる．ここで重要なのは直前の 35 分経過時の処理時間も明らかではないが，少し遅くなっているということだ．基本的に約 20ms 以内で処理が終わっているのに対し，40ms ほどかかっている．

パフォーマンス障害の性質として，徐々に性能が低下して発生するというものがある．そのため，最初に性能が低下したタイミングをつかむのが重要である．つまり，明らかな遅延が発生したタイミングから少しずつさかのぼって解析をしていく必要がある．そこで，明らかな遅延の前段階の少し遅れた処理も，ログに残しておくことで，さかのぼって調べることができるようになり，手がかりを増やすことに繋がるのである．要するに，ある程度の量のログを一時的に保存しておき，明らかな障害が発生した時に，それらもまとめてログに記録するということを行う．なお，40 分経過後の処理時間も遅くなっているように見えるが，これはパフォーマンス障害が発生した結果，その影響により蔓延的に遅延が発生しているだけである．そのため，直後に関するログは取得しない．

上記の理由により,，本稿で提案する手法は異常が検出された時点より前のログも残しておくようにする．このようにすることによって，障害発生の段階から根本原因をたどるための手がかりが得やすくなると期待できる．なお，どの程度前までのログを残しておくべきかという点については状況に応じて慎重に決める必要がある．できる限り多くのログを残しておけば根本原因究明のための手がかりが得やすくなる一方で，ログ容量はほとんど削減されない．一方，ログ容量を削減するために残しておくログを減らせば，根本原因究明のための重要な手がかりが失われてしまう可能性が高くなる．根本原因の発生から障害発生までに生成されるログの容量は一定ではないため，システムに許容されるログの容量に応じて適宜，決めていく必要がある．しかし，管理図を用いることで，限られた容量であっても より有益と思われるログが取得できるという利点がある．

なお，この方法はシステムに依存しないため Linux だけでなく様々なシステム（FreeBSD，Windows など）にも対応できる．そのため，導入が簡単であるという利点がある．

4.3 実装

4.3.1 全体像

図 6 が，本稿で実現する管理図を用いたロギングのイメージである．既存のロギング方法では，あらゆるログデータが記録されていく．もちろんパフォーマンス障害において，詳細な動作履歴を残すのは，必要である．しかし，それによる CPU 利用率，ログサイズの増大が問題になると述べてきた．そこで，本稿ではその中でもより重要だと思われるログのみを記録する．本来あらゆるログが記録されるものを，管理図を用いてフィルタリングすることで，必要なログデータのみを取得する．必要なログデータとは，すでに文献 [2] で説明したように，処理時間が異常に遅く

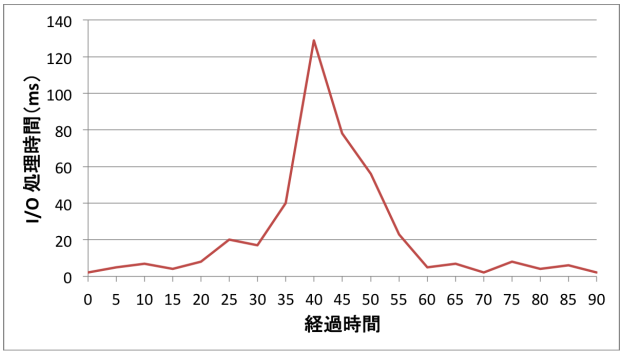

図 5 パフォーマンス障害例

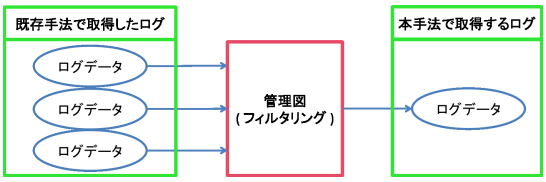

図 6 全体像

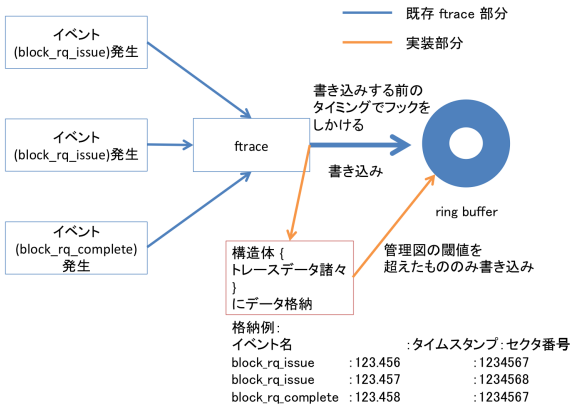

図 7 実装部分

なっているものに関するログである．この異常と判定された処理に関するログを必要と判定し取得する．これにより重要性が高いログのみが取得され，ログ削減に繋がる．

4.3.2 実装部分

図 7 に実装した部分を示す．ログの取得には Linux カーネルに標準で搭載されるトレースツール ftrace [4] を用いた．また実装したカーネルバージョンは Linux 2.6.34.7 である．これは対象とする障害事例が発生したカーネルバージョンが Linux 2.6.34.7 だからである．図 7 の青い矢印の処理が既存 ftrace 部分，オレンジの矢印の処理が本稿で述べる実装部分である．

まず ftrace の実装上の仕組みについて説明する．図 7 の青い矢印の処理が ftrace のログ取得に関する処理の流れである．ftrace はなんらかの I/O 処理が発生した時，それに応じたイベントログを発行する．表 1 にイベントの例と，それが発行されるタイミングを示した．I/O 処理に関するイベントをピックアップした．イベントは他にも 1000 種類以上あり，取得するイベントは自分で設定することができる．こうしたイベントが発生すると，ftrace はそれに関するログをリングバッファに書き込むようになっている．リングバッファとは，あらかじめ決めたサイズの範囲内で記録を行い，それを超えた場合古いものが新しいものによって上書きされるという仕組みの記録領域である，このサイズは設定することができる．本稿では 512MB に設定している．つまり，ftrace は I/O 処理が発生すると，それに応じたイベントが発生し，それに関するログをリングバッファに書き込むという流れになっている．

本稿ではログ削減を行うため，リングバッファに書き込まれる量を削減しなければならない．そこで，リングバッファに書き込みを行う直前で管理図によるフィルタリングを行う．図 7 のオレンジ色の矢印のように処理を加えることで，ftrace がリングバッファに書き込む直前のタイミングにフックをしかけ，実装する．実装する処理として，おおまかに分けて二つ，特性値の測定，管理図によるフィルタリングを行うことで実現する．なお，本稿で述べるプロトタイプシステムでは，ftrace の問題で実装が難しく，障害発生前のログ取得機構はまだ実装できていない．

4.3.3 特性値の測定

まず一つ目の処理，特性値の測定について説明する．特性値とは，すでに文献 [2] で説明したとおり，管理図において監視対象とする値のことである．本稿で述べるプロトタイプシステムでは，対象とする障害事例が I/O に関するものであるため，I/O 処理時間に着目する．特にスタベーション，ブロックなどにより遅延が起きやすいキューにおける滞留時間を特性値として用いる．

次のようにして，キューの滞留時間の計測を行なう．ftrace は前節で説明したとおり，イベントが発生すると，それに関するログを記録する．ftrace が記録するログは図 9 に示したようなものである．プロセス名，タイムスタンプ，イベント名，セクタ番号といった情報が書き込まれている．本稿では，I/O 処理時間，特にキューの滞留時間に着目すると述べてきた．そこで，キューの滞留時間を測定する．キューの滞留時間は同じセクタ番号に対する 2 つのイベント間，block_rq_issue 発生から block_rq_complete 発生までの時間ということが，調査により分かっている．そこで，この 2 つのイベントに着目する．

処理の流れを図 8 に示した．「リストに挿入」，「処理時間の計算」という部分が特性値の測定の処理である．着目する 2 つのイベント block_rq_issue，block_rq_complete のどちらが発生するかに応じて，処理を変える．

まず block_rq_issue 発生時の処理を説明する．このイベントが発生した場合，ログの情報 (タイムスタンプ，イベント名，セクタ番号) を一時的にリストに挿入するという処理を行う．例えば，図 9 を用いて説明すると，まず上から二行目のログが block_rq_issue 発生に対応するログである．こうした block_rq_issue に対応したログが発生した際に，そのログのタイムスタンプ，イベント名，セクタ番号の 3 つの情報をリストに挿入するということである．これで block_rq_issue 発生時の処理は終わりである．つまり，タイムスタンプ，イベント名，セクタ番号の 3 つの情報を保持するリストが作成されていく．

次に block_rq_complete 発生時の処理を説明する．このイベントが発生した場合，先ほど説明した block_rq_issue 発生時の処理により作成されたリストに対し，処理を行う．本稿において，特性値はキューの滞留時間であると説明した．また，キューの滞留時間は同じセクタ番号に対する block_rq_issue 発生から block_rq_complete 発生までの時間であると説明した．そこで，行う処理として，リストから同じセクタ番号に対する block_rq_issue 発生時のログ情報を検索する．block_rq_complete 発生時には，それが

表 1 ftrace イベント例と説明

イベント名	説明
block_bio_queue	I/O 要求が発行される時に発生
block_rq_complete	I/O 要求が完了した時に発生
block_rq_requeue	I/O 要求が再度キューに入る時に発生
block_rq_issue	I/O 要求がキューに入る時に発生

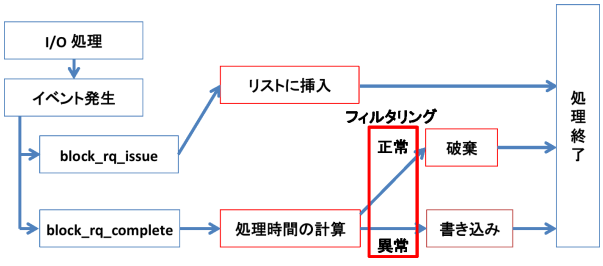

図 8 処理の流れ

```
sample-30291,423021.983432,block_bio_queue,129685415
sample-30291,423021.983437,block_rq_issue,129685415
<idle>-0,423021.990683,block_rq_complete,129685415
sample-30291,423021.990747,block_bio_queue,129697928
sample-30291,423021.990757,block_rq_issue,129697928
```

図 9 ftrace が記録するログデータの例

対象とするセクタ番号がわかるので，それをキーにリストを検索する．セクタ番号でリストを検索した結果に対応するログの情報から block_rq_issue 発生時のタイムスタンプを得ることができる．その値と，block_rq_complete 発生時のタイムスタンプとの差分を計算することで，キューの滞留時間を求める．例えば，図 9 でセクタ番号 129685415 の block_rq_issue から block_rq_complete までの時間を計算した場合，0.007246 ms ということがわかる．この値が特性値であり，これを管理図の監視対象とする．なお，リストを検索し，ペアが見つかり次第，その情報はリストから削除していく．これにより，リストが永遠に長くなることはなくなる．

4.3.4 管理図によるフィルタリング

測定した特性値を用いて，管理図によるフィルタリングを行う方法について，説明する．まず，管理限界線は次のようにして決める．前節で述べた計測方法により特性値を得るが，この特性値を 100 個取れるまで記録していく．100 個取った後，5 個ずつ 20 の群に分類する．そして文献 [2] で述べた方法により，管理限界線を求める．

次に求めた管理限界線と特性値を比較する．比較は，block_rq_complete 発生時に特性値を測定した後のタイミングで行う．この比較により，測定した特性値が管理限界線を上回っている場合にのみ，ftrace はリングバッファへの書き込み処理を行う．それ以外の場合には，そのログ情報を書き込まずに，破棄する．これにより，ログを削減する．

管理図の UCL を決めるタイミングだが，出荷前のテストの最終段階で，正常に動作していると思われる段階で決めておく．この段階で異常値が発生しているものは出荷しないので，正常に動作していると仮定して UCL を定めて問題ないと思われる．

また，システムの変化，劣化などにより，性能の期待値が低下することがある．例えば，リリース後初期の段階とリリース後 10 年経過時の段階では，システムに期待される性能値が全く違うというのは容易に想像できるだろう．そういった場合にも対処できるように，管理限界線は動的に再計算し変更できるようにした．動的に変更するタイミングについては，システムに応じてシステムのバージョン変更時なのか，一定の期間ごとなのか適宜決める必要があるが，状況に応じて管理限界線を変更することで，システムの期待性能自体が劣化した状況での処理を異常と判定しないようにできる利点がある．また，この再計算はシステムの稼働を止めることなく，行うことができる．

以上の方法により，管理図によるフィルタリングを行い，異常と判定された場合のみ ftrace がリングバッファへの書き込みを行うという処理を実現した．

5. 評価実験

本章では，提案手法が実際にパフォーマンス障害に対し，ログを削減できるかの検証を行った．具体的には，1) 実際の障害事例を用いたときの，ログの削減量，2) 既存 ftrace との CPU 利用率の比較 について実際に本手法を適用した結果を検証した．検証を行う環境は，RedHat Enterprise Linux 5.4 および 6.4[7]，カーネルのバージョンは Linux-2.6.34.7, メモリは 16 GB 1333 MHz DDR3 でコア数が 4, プロセッサは Intel® Xeon® CPU E3-1270 @ 3.40 GHz のマシンを使用した．また，ディスクは ext3 にフォーマットした 80GB のパーティションを用いた．

5.1 障害事例

本実験では，国内 IT ベンダの社内インフラ系サーバ内で発生した実際のパフォーマンス障害の事例を対象として検証を実施した．本項では事例の説明を行っていく．

本事例は，RedHat Enterprise Linux 5.4 および 6.4 で発生したパフォーマンス障害である．事例の現象としては，1 つのディスクに対して 300 超のプロセスが同時に read 命令を行い続けると，一部の read に 7 秒の遅延が生じるという現象である．この障害の発生条件として，次の 2 つの条件がある．1 つ目は特定のブロックデバイスに多数のプロセスが同時に同期 I/O を行い続けるという点である．2 つ目の条件はブロックデバイスの I/O スケジューラとして CFQ スケジューラを使用するというものである．

CFQ スケジューラとは Linux に標準で搭載されている I/O スケジューラである．CFQ スケジューラでは，プロセス毎に I/O の優先度をつけることができ，各プロセスに対応するキューにはプロセスの I/O 優先度に応じた「持ち時間」が割りあてられている．各リクエストキューはラウンドロビン方式で順に選択され，選択されたリクエストキュー内の I/O リクエストは持ち時間を過ぎるまで次々とデバイスのキューへと送られる．優先度に応じて持ち時間を定めることによって，優先度の高いプロセスの I/O が優先的にスケジュールされるようになっている．

5.2 実験方法

パフォーマンス障害を意図的に発生させ，障害環境を構築する．既存 ftrace でログを取得する場合と，管理図適用後の ftrace でログを取得する場合の 2 パターンで実験を行う．管理図適用後の ftrace でログが削減できるか検証す

る．また，CPU 利用率について，既存 ftrace と管理図適用後 ftrace を比較する．

5.3 実験結果

5.3.1 ログの削減

図 10 に既存 ftrace により記録されたログサイズ，管理図適用後 ftrace により記録されたログサイズを示した．横軸は実験の経過時間，縦軸はログサイズである．なお，既存 ftrace が記録対象としたイベントは，管理図適用後 ftrace が記録対象とするイベントと同様の設定 (block_rq_issue, block_rq_complete を記録) にした．そのため，すでにログのサイズを絞り込んだ設定との比較である．それと比較しても，約 160MB のログを 約 14MB と，約 1/10 のサイズに削減することができた．また，今回の実験では動作プロセス数が少ないタイミングで管理限界線を定めたため，特性値が小さく，緩い基準線での結果となった．管理限界線を決めるタイミングを慎重に議論することで，さらなるログの削減が見込めるはずである．

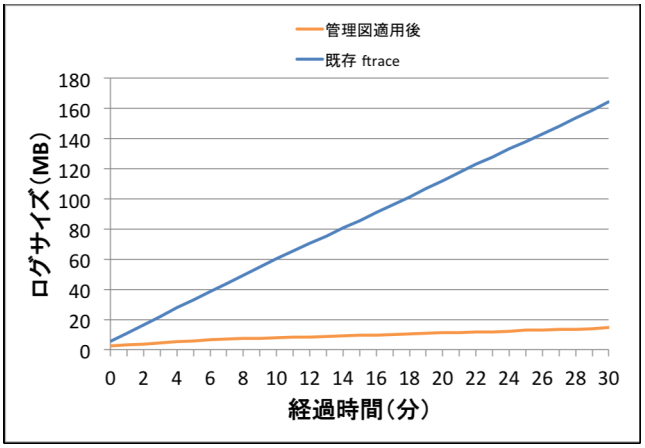

図 10 ログサイズの比較

5.3.2 CPU 利用率の比較

既存 ftrace, 管理図適用後 ftrace について I/O wait CPU の割合，Idle CPU の割合を比較した．管理図を適用し，ディスクへの書き込み処理が減ることで，I/O wait が減っているかを検証する．また正常値に関するログの処理がなくなることで，CPU の負荷が減っているかを検証する．

図 11 に I/O wait CPU の割合を示した．横軸は実験の経過時間，縦軸は I/O wait 状態となっている CPU の割合である．実験の経過時間全体を通して，既存 ftrace に比べ管理図適用後 ftrace の方が I/O wait CPU の割合が少ないことがわかる．これは，管理図により必要なログの書き込みのみが行われるようになったため，余分な I/O 処理が減り，結果として I/O wait が減ったということである．以上から，管理図を適用したことで必要な I/O 処理だけが行われるようになったと言える．

図 12 に Idle CPU の割合を示した．横軸は実験の経過時間，縦軸は Idle 状態となっている CPU の割合である．実験の経過時間全体を通して，既存 ftrace に比べ，管理図適当後 ftrace の方が Idle CPU の割合が多いことがわかる．これは，管理図を適用したことで，正常値に関するログの処理がなくなり，CPU の負荷が減ったためである．実験の後半で両者の値が近づいているのは，管理図適用後でも，処理が激しくなると，マシンに負荷がかかり，異常と判定される処理が増え，ログの書き込み処理が増えるためである．以上から，管理図により CPU の負荷が減っていることがわかった．

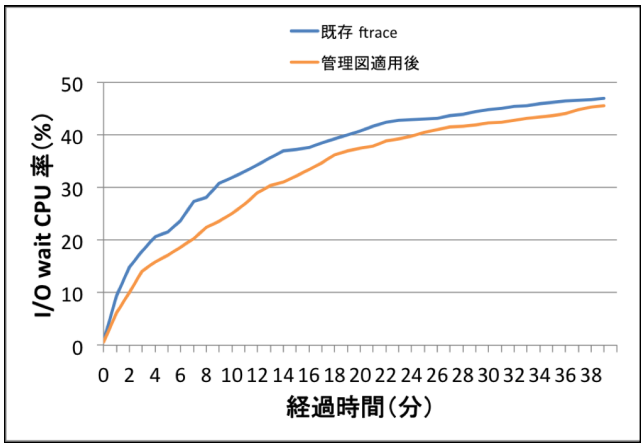

図 11 I/O wait CPU の割合

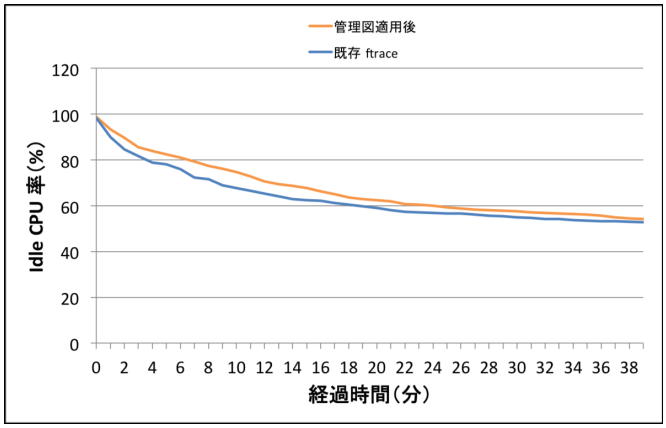

図 12 Idle CPU の割合

6. 関連研究

Yuan 達は, ログに出力する情報を増やすことによって, 詳細な調査ができるようにしている [8]. この研究ではログの情報量を増やすため, 短いサイクルでのログの取得に向いている. 一方, 本稿では余分なログ情報を排除することによってメモリへの負荷を下げ, 長いサイクル上でのパフォーマンス障害に対しても必要なログ情報を得ることができる.

また, ログに関連した研究として Xu 達はソースコードを静的解析し, コンソールログを詳細に分析することで, 障害検知に活かそうという研究をしている [9]. この研究は, ソースコードの情報を得ることができるシステムには向いている. しかし, 本稿で提案する手法はソースコードの情報を必要とせず, 簡単に導入することができる.

Attriyan 達は, 重み付けを行う事によってパフォーマンス障害の原因を推測する X-ray というツールを開発している [10]. X-ray はエンドユーザの設定ミスを原因とするパフォーマンス障害を対象としている.

Montesinos 達は, チャンクサイズを増やしログ取得を行うエントリを減らすことで, ログの削減をしている [11]. 本稿とは異なり, こちらはパフォーマンス障害に着目しているものではない.

7. 結論

本論では, 管理図によるログの削減手法を提案した. 提案手法は管理図という統計的手法を用いることで, 障害に関わっている部分のログは残しつつ, 全体のログの取得量を削減するという手法である.

評価実験として, 国内 IT ベンダから提供を受けた障害事例に対して本手法を適用した. その結果, ログを約 1/10 のサイズに削減することができた. また, CPU 利用率に関して, 既存 ftrace と本手法を比較した. 既存 ftrace に比べ, I/O wait CPU の割合は減り, Idle CPU の割合は増えていたことから, 無駄な処理を減らすことができたことが示された.

今後の課題として, 管理限界線を決めるタイミングの検討, 他の障害事例に対しても効果があるか確認することが挙げられる.

参考文献

[1] Networkworld. http://www.networkworld.com/news/2012/102212-amazon-ebs-263592.html.
[2] 深井 賢, 中村 啓太郎, 吉村 剛, and 河野 健二. Linux における障害異常検知への管理図の適用. In **研究報告システムソフトウェアとオペレーティング・システム（*OS*）**, pages 1–7, July 2014.
[3] Shewhart control chart. JIS Z 9021.
[4] ftrace. https://access.redhat.com/site/documentation/ja-JP/Red_Hat_Enterprise_Linux/6/html/Developer_Guide/ftrace.html.
[5] Lloyd S Nelson. The shewhart control chart–tests for special causes. In *American Society for Quality Control (ASQC '84)*, pages 237–239, October 1984.
[6] Lloyd S Nelson. Interpreting shewhart x-bar control charts. In *American Society for Quality Control (ASQC '84)*, pages 114–116, April 1985.
[7] redhat. www.redhat.com/.
[8] Ding Yuan, Jing Zheng, Soyeon Park, Yuanyuan Zhou, and Stefan Savage. Improving software diagnosability via log enhancement. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems (ASPLOS '11)*, pages 3–14, March 2011.
[9] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP '09)*, pages 117–132, October 2009.
[10] Mona Attariyan, Michael Chow, and Jason Finn. X-ray: Automating root-cause diagnosis of performance anomalies in production software. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation (OSDI '12)*, pages 307–320, October 2012.
[11] Pablo Montesinos, Luis Ceze, and Josep Torrellas. Delorean: Recording and deterministically replaying shared-memory multiprocessor execution efficiently. In *Proceedings of the 35th Annual International Symposium on Computer Architecture (ISCA '08)*, pages 289–300, May 2008.