Vol.2019-OS-147 No.15
2019/7/25

Pause Loop Exit の多発につながる KVM における仮想 CPU スケジューリング

安野 直樹[1]　石黒 健太[1]　河野 健二[1]

概要：
仮想化環境では物理 CPU の個数よりも多くの仮想 CPU を利用することが一般的になっている．しかし，ゲスト環境は仮想 CPU がプリエンプトされることは想定せずに実装されている．そのため，仮想 CPU のプリエンプトによりスピンロック中などに仮想時間がとぎれると Pause Loop Exit (PLE) というイベントが発生し，仮想マシンモニタに通知が行われる．KVM には PLE が発生すると，スケジューラにヒントを与えることで PLE の多発を避ける機構がある．しかしながら，現状の KVM ではこの機構が十分に機能しておらず， PLE が多発することがある．
本研究では PLE が多発する原因を分析する．その結果，PLE の多発は 1) 与えられたヒントよりも fairness を優先してスケジューリングしている，2) 優先実行すべきとヒントで与えられた仮想 CPU が PLE が起こした仮想 CPU とは別のランキュー上に存在している，という 2 つが原因で，スケジューラに与えたヒントが上手く活用されていないためだと示す．また，それらを解決するための手法を提案する．

キーワード：
クラウド・コンピューティング，仮想化，Pause Loop Exit

1. はじめに

複数の仮想マシンを一つの物理マシン上に統合することはクラウドコンピューティングなどにおいてマシンリソースの利用効率を高める手法として広く用いられている．そのような環境では，物理 CPU の数を超える仮想 CPU を作成して 1 つの物理 CPU を複数の仮想 CPU が共有する．その結果，仮想 CPU の実行がプリエンプティブ・スケジューリングにより，時間的に不連続となる．この時間的不連続性により，**仮想時間不連続性問題** (*virtual time discontinuity problem*)[1] という様々な問題が発生することが知られている．しかしながら，ゲストオペレーティングシステム（ゲスト OS）は CPU が常時稼働しているという前提で設計されているため，仮想 CPU のデスケジュールにより想定外のブロッキングが発生する．仮想時間不連続性問題とは，この想定外の仮想 CPU ブロッキングにより発生するゲスト OS の著しい性能低下である．

仮想時間不連続性問題の一例は，過度に長いビジーウェイトの発生である．ゲスト OS では短時間で終わるクリティカル・セクションの保護にはスピンロックを用いることが一般的である．しかし，クリティカル・セクションを実行している仮想 CPU がプリエンプトされている場合，その仮想 CPU がスケジューリングされるまでスピンロックの解放が行われない．その結果，スピンロックを獲得しようとする他の仮想 CPU が長時間，ビジーウェイトすることになり CPU 時間が無駄になりパフォーマンスの低下につながる．

過度に長いビジーウェイトを減らすためのハードウェア機構として，Intel 社から Pause Loop Exiting（PLE）[2]，AMD 社から Pause Filtering（PF） [3] が提供されている．これらは一定時間以上ビジーウェイトをしている仮想 CPU を検知し，制御を仮想マシンモニタ (VMM) に移すものである．VMM に制御が移ると，仮想 CPU のスケジューリングをやり直すことができる．この時，VMM は PLE の要因となったビジーウェイトを取り除くように適切な仮想 CPU スケジューリングを行う必要がある．上記の例であれば，クリティカルセクション内で停止している仮想 CPU をスケジュールするのが望ましい．PLE と

[1] 慶應義塾大学
Keio University

PF の提供する機能は実質的に同じであり，以降は両者を区別せず PLE と呼ぶ．PLE を利用した場合でも，長時間のビジーウェイト発生時に VMM に制御が移るに過ぎず，PLE の要因となったビジーウェイトを取り除く仮想 CPU スケジューリング手法が課題となっている．仮想 CPU スケジューリングが適切に行われないと PLE が多発し，十分に機能を発揮しない．

Kernel-based Virtual Machine(KVM)は Linux に VMM としての機能を追加する Linux サブシステムである．[4] KVM は PLE に対応したスケジューリングを行なっており，PLE 発生時にその要因を取り除くために優先的に実行すべき仮想 CPU を推測し，優先度をブーストする機構がある．しかし，そのような対策を行なっているにも関わらず，KVM では PLE が多発していることが知られている．[5] そのため，現状の KVM の PLE の多発対策は不十分であると言える．

本稿では，KVM の仮想 CPU スケジューリングが PLE の多発に繋がる原因を調査する．その結果，以下のケースで PLE の多発に繋がる，極めて非効率的なスケジューリングが行われることを明らかにした．

- PLE の要因を取り除くために優先的に実行すべき仮想 CPU を適切に選べない時．
  - KVM はカーネルモードの仮想 CPU のみを優先度ブースト対象としているため，PLE の要因を取り除くために優先実行すべき仮想 CPU がノーマルモードの時は適切な仮想 CPU の優先度をブーストできない．
- 優先度がブーストされた仮想 CPU が，実際には優先的にスケジューリングされない時．
  - KVM は仮想 CPU 間の CPU 時間の fairness が崩れるようなスケジューリングは行わない．
  - KVM は PLE を起こした仮想 CPU と優先度ブーストした仮想 CPU の存在するランキューが違う時，優先度ブーストした仮想 CPU が実行されるまで時間がかかる．

上記の分析の基づき，その対策としてカーネルモードの仮想 CPU をブースト対象に加える手法，fairness を無視したブーストを行う手法，そして優先度ブーストを行なった仮想 CPU を PLE を起こした仮想 CPU のランキューへ移動させる手法の 3 つを提案し，最初の 2 つの手法を評価する．結果，最大 60%以上の性能向上するケースがあることがわかった．

本論文の構成を示す．2 章では仮想時間の不連続性について説明する．3 章では KVM の仮想 CPU スケジューリング手法について説明する．4 章では分析環境及びその結果を示す．5 章では調査を元にした対策手法を提案する．6 章では本提案の有効性を評価する．7 章では関連研究について述べる．8 章では本論文のまとめを述べる．

2. 仮想時間の不連続性

1 つの仮想 CPU を複数の仮想 CPU が共有すると，仮想 CPU の実行は不連続になる．これは，仮想 CPU が物理 CPU の数を超えている場合，仮想 CPU のスケジューリングが行われ，プリエンプトされて仮想 CPU の実行は停止する．そのため，実時間でみると仮想 CPU の実行は不連続となる．

2.1 仮想時間不連続性問題

仮想時間の不連続性により，ゲスト OS が本来意図していない挙動を引き起こし，仮想マシンの性能が大幅に低下することがある．これを仮想時間不連続性問題という．このような問題は，CPU が常時稼働しているというゲスト OS の設計の前提が崩れるために起こる．

仮想時間不連続問題の本来意図しない挙動の一つとして，ビジーウェイトが過度に長くなるという現象がある．ゲスト OS は短時間で終了する処理の終了を待つ際はビジーウェイトする．しかしながら，短時間に終わるはずの処理を行なっている仮想 CPU がプリエンプトされると，いつまでもビジーウェイトが完了することができない．

ビジーウェイトが過度に長くなる例として，Lock Holder Preemption (LHP) 問題 [6] が挙げられる．図 1 は LHP が起こる様子を示したものである．スピンロックを確保した仮想 CPU がプリエンプトされた場合，ロックが長時間解放されず，そのロックを獲得しようとする他の仮想 CPU が長時間ビジーウェイトする．

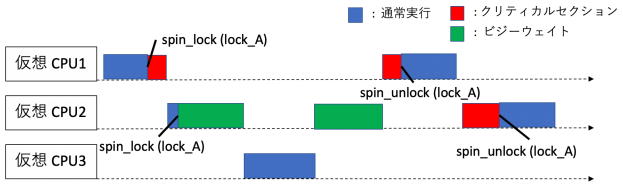

図 1 LHP 発生の様子

2.2 Pause Loop Exit(PLE)

ビジーウェイトが過度に長くなる問題を緩和する手法として，Intel 社では Pause Loop Exiting (PLE) というハードウェア機能が提供されている．PLE とは，x86 においてビジーウェイトを行う際は PAUSE 命令を繰り返し実行するように推奨されていることを利用し，一定以上の PAUSE 命令を検知して VMM に制御を移す機能である．これにより，過度に長いビジーウェイトを起こしている仮想 CPU を検知し，他の仮想 CPU の再スケジューリングが可能になる．

PLE を利用するには，PLE_Gap と PLE_Window という 2 つのパラメータをあらかじめ設定する．図 2 の通り，PLE_Gap は PAUSE 命令が連続で実行されていると認識するまでの最大時間で，PLE_Gap よりも短い周期で PAUSE 命令が PLE_Window 以上の時間連続すると PLE が発生する．

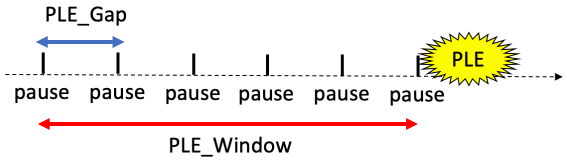

図 2　PLE の起こる様子

PLE が発生した時，その要因を取り除かないと PLE を起こした仮想 CPU が再度スケジューリングされる度に何度でも PLE が再発し，PLE の多発へ繋がる．PLE が多発すると，VM exit に伴うコンテキストスイッチに加え，ビジーウェイトによる CPU 時間が浪費される．よって，PLE 発生時に素早くその要因を取り除くよう適切に仮想 CPU をスケジューリングする必要がある．例えば，LHP 問題による PLE の場合，スピンロックを保持したままプリエンプトされた仮想 CPU を優先的にスケジューリングすることで，スピンロックが解放され，PLE の要因を取り除くことができる．

3. KVM のスケジューリング

KVM は Linux のデフォルトスケジューラである CFS（Completly Fair Scheduing）に基づいて仮想 CPU をスケジューリングする．CFS はスレッドを基本単位としたスケジューリングであり，KVM では仮想 CPU をスレッドとして扱う．CFS はスレッドごとに優先度に応じた重み付けを行なったのち，消費した仮想 CPU が等しくなるよう CPU 時間を割り当てる．

Linux ではキャッシュローカリティのために，各スレッドは物理 CPU どれかに紐づいたランキューに割り振られ，紐づいた物理 CPU 上で実行される．どのランキューに配置されるかは一定時間ごとにロードバランサにより変わることがある．さらに，ランキュー中の特定のスレッドを高優先度でスケジューリングするようスケジューラにヒントを与える機構もある．ただし，対象のスレッドを実行することでスレッド間の CPU 時間の fairness が崩れる場合は，fairness が保てると判定されるまでは実際には実行されない．

3.1 KVM の PLE への対応

現状の KVM は，PLE 発生時には Candidate VCPU selection[7] という仮想 CPU スケジューリング方法を用いて PLE の多発を避ける仕組みが備わっている．これはスピンロックの獲得待ちを要因とした PLE の発生を念頭に，スピンロックを獲得したままプリエンプトされた可能性のある仮想 CPU の優先度をブーストするというものである．Candidate VCPU selection はプリエンプトされている仮想 CPU を Resource-waiter と Lock-waiter の 2 種類に分類する．Resource-waiter はタイムスライスを使い切ったことが原因でプリエンプトされている仮想 CPU で，Lock-waiter は PLE が発生したことでプリエンプトされている仮想 CPU である．仮想 CPU は仮想マシンごとに，図 3 のように円状のリストで管理される．PLE の発生時には，最後に優先度をブーストされた仮想 CPU を起点としてこの円状リストを辿り，最初の仮想 CPU の優先度をブーストする．ただし，Lock-waiter の仮想 CPU は check をつけておくだけとし，全ての仮想 CPU が Lock-waiter のためどの仮想 CPU もブーストされずに 2 周目に差し掛かった場合のみ，check のついた Lock-waiter をスケジューリングする．これは，Lock-waiter よりも Resource-waiter の方が処理を進めることができる可能性が高いと期待されるため，Resource-waiter を優先的にスケジューリングするためである．

また，KVM はカーネルモードの仮想 CPU のみをこの優先度ブースト対象としている．これは KVM が PLE の発生要因を LHP 問題によるものだと仮定しており，スピンロックは一般的にカーネル空間でしか使われないためである．

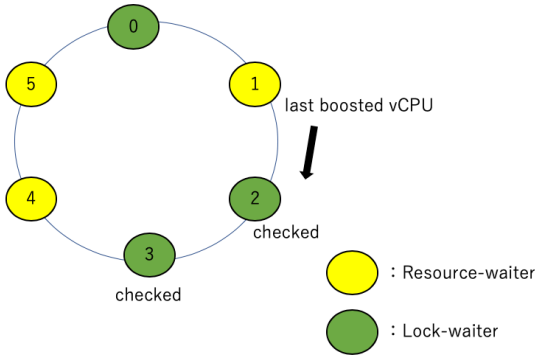

図 3　PLE 時の KVM の仮想 CPU スケジューリング

しかしながら，KVM には PLE の多発を避ける仕組みがあるにも関わらず，我々の調査では PLE が多発してい

ることが確認できた．図 4 は各仮想 CPU の PLE の連続発生回数をトレースポイントから調査したものである．実験環境は 6 章の通りである．例えば，dbench では 50%以上の割合で PLE が 53 回以上連続しており，最大で 522 回連続的に発生している．Candidate VCPU selection はラウンドロビンを基本としており，2 周するまでにブースト対象となる仮想 CPU が最低一回は優先度ブーストされるようになっている．この実験環境である仮想 CPU 数 4 つに対して PLE がこれだけ多く連続的に起こることは考えにくく，本来意図していない極めて非効率的なスケジューリングが行われていると言える．

4. KVM における PLE の多発要因の分析

この章では，KVM において PLE が多発する原因を分析する．KVM における PLE の発生要因はスピンロックの待機待ち，プロセッサ間割り込み（IPI）におけるバリア同期，tsc を用いた待機の 3 つである．[5] 本稿ではスケジューラにより対策困難な，tsc を用いた待機は対象外とする．また，PLE Window は適切な値，つまり本来は通常のビジーウェイトが誤って過度に長いと誤検知されて起こった PLE はないものとする．これを元に，PLE の要因を長時間取り除けないスケジューリングが行われるケースについて考える．

まず，優先度ブーストの対象仮想 CPU を適切に選択できないケースについて考える．KVM において，ノーマルモードの仮想 CPU は優先度ブーストの対象から外れる．これは，PLE は LHP 問題によって起こるという前提を元にしており，スピンロックは一般的にカーネル空間でのみ使用されるためである．しかしながら，PLE は スピンロックの待機待ちの他，IPI におけるバリア同期でも起こる．バリア同期時の PLE は IPI の送り先を実行して IPI 処理を進めることで，その要因を取り除くことができる．しかし，KVM は IPI のバリア同期による PLE を想定しておらず，IPI の送り先がノーマルモードの時はいつまでも送り先が優先実行されない．

次に，PLE の要因を取り除くために優先的に実行すべき仮想 CPU を適切に選択して優先度をブーストしているのに，その仮想 CPU が実際には優先的に実行されない場合について考える．KVM のスケジューラは Linux のデフォルトスケジューラである CFS スケジューラである．CFS は仮想 CPU を普通のスレッドとして扱うため，CPU 時間の fairness を仮想マシン間ではなく仮想 CPU 間で取っている．また，PLE を起こした仮想 CPU と優先度ブーストを行なった仮想 CPU の存在するランキューが違う時，優先度ブーストを行なったランキューから現在実行されているスレッドの実行が停止するまでスケジューリングが行われず，優先度が高くても即座には実行されない．

5. 提案手法

この章では 4 章の分析結果を元に，KVM において PLE の多発を避けるための手法を提案する．

まず，優先度ブーストすべき仮想 CPU を正しく選べないケースは，カーネルモードの仮想 CPU を選択肢から外していることが原因である．よって，ノーマルモードの仮想 CPU を優先度ブーストの対象に加える手法を提案する．これにより，IPI の送信先がノーマルモードの時も PLE の要因を素早く取り除くために優先度ブーストを行うべき仮想 CPU を正しく選ぶことが可能である．

次に優先度ブーストをした仮想 CPU が実際には優先的にスケジューリングされないケースについて考える．この要因の 1 つ目である，スケジューラに与えたヒントよりも fairness を優先してしまうことについては，優先度ブーストされた仮想 CPU のスレッドに関しては仮想 CPU 間の fairness が崩れても，仮想マシン間の fairness が保証されていれば実行してしまう手法を提案する．仮想環境を提供する VMM にとって重要な fairness は仮想 CPU 間ではなく，仮想マシン間である．そのため，PLE を起こして処理を進められない仮想 CPU へ与えられるはずの CPU 時間を他の仮想 CPU へ分け与えても fairness の観点上大きな問題はなく，パフォーマンスの観点から有益であると言える．2 つ目の要因である，PLE を起こした仮想 CPU と優先度ブーストを行なった仮想 CPU の存在するランキューが違う場合については，優先度ブーストをした仮想 CPU を PLE を起こした仮想 CPU のランキューへと移動させる手法を提案する．

6. 評価

この章では，5 章で提案した手法のうち，ノーマルモードの仮想 CPU を優先度ブーストの選択肢に加える手法，および優先度がブーストされた仮想 CPU は fairness を無視してブーストする手法の有効性について評価を行う．ただし，fairness を無視するケースについては特定の仮想 CPU が余分に得た CPU 時間の分だけ同一仮想マシン上の他の仮想 CPU の CPU 時間を減らすべきであるが，CFS スケジューリングでは一時的に特定のスレッドが多く CPU 時間を持とうとも，時間とともに平滑化されるため，実装の簡単さを優先してそのような機構は設けないものとする．

6.1 実装と実験方法

本調査において，Intel Xeon CPU X5650 2.67GHz 6 コア，メモリ 4GB のマシンを使用した．ホストのオペレーティングシステムには Ubuntu 18.04LTS をインストールし，Linux kernel 4.15.18 をビルドした．VMM は KVM を使用した．ゲストのオペレーティングシステムには Ubuntu

14.04LTS をインストールし，Linux kernel 4.4.134 を使用した．ゲストマシンを 2 つ作成し，それぞれ仮想 CPU を 4 つ，メモリは 1GB ずつ割り当てた．また，PLE_Window の動的決定を無効化し，過度に長い訳ではないビジーウェイトで誤検知により PLE が起きないよう，十分に大きい値である，KVM のデフォルト値である 2048 cycle の 16 倍の値を用いた．

ベンチマークはマルチスレッドベンチマークである ebizzy[8], PARSEC[9], hackbench[10], dbench[11], pbzip2 [12] を使用した．1 つの仮想マシン上で PARSEC の swaptions を実行し，もう一方の仮想マシン上で評価するベンチマークを実行した．

6.2 PLE の連続

まず，各仮想 CPU ごとの PLE の連続数をベンチマークごとに 1 秒間測定した結果である．ここで言う連続とは，PLE を起こした仮想 CPU を次回スケジューリングした時，再度 PLE を起こしたことを意味する．PLE が連続した時，PLE を引き起こした要因を取り除けないまま再度その仮想 CPU が実行されたことを意味する．そのため，連続数は多い方がより非効率的なスケジューリングであると言える．図 4 はその結果である．

ebizzy と pbzip2 は特に TLB shoot down を多く発生させるベンチマークであり，主に IPI 処理のバリア同期中に PLE を起こす．このようなワークロードでは，ノーマルモードの仮想 CPU を優先度ブーストする手法により連続数が減っている．しかし，ノーマルモードの仮想 CPU をブースト対象に加えないまま fairness を無視したブーストを行うと，PLE の要因を取り除けない誤った仮想 CPU をより激しくブーストしてしまう．そのため，より連続数が長くなっているのだと推測できる．逆に，hackbench は主にスピンロックの待機待ち中の PLE を起こす．そのため，ノーマルモードの仮想 CPU をブースト対象とすると，連続数が悪化している．しかし，fairness を無視したブーストを行うことで，連続数が大幅に改善されていることがわかる．

6.3 パフォーマンス

本提案手法を用いた場合のパフォーマンスの向上について評価を行なった．結果を対策なしを元に正規化したものが図 5 である．数値が大きいほど高いパフォーマンスであることを示している．

ebizzy に関して，大幅な性能向上が見られて．二つの手法を組み合わせた時パフォーマンスは最も高くなり，対策なしに比べて 163%向上している．一方，fairness を無視したブースト単体ではあまり効果がないことがわかる．これは，ebizzy は IPI におけるバリア同期が主な PLE の要因であるため，ノーマルモードの仮想 CPU がブースト対象とならないと，そもそもブーストすべき正しい仮想 CPU を選択できないためだと考えられる．

逆に，hackbench はカーネルモードの仮想 CPU をブーストすることで，パフォーマンスが約 2.3%と低下している．これは，hackbench はスピンロックの待機待ち中の PLE が多く，そのような PLE ではスピンロックのないノーマルモードの仮想 CPU を実行することは PLE の多発へ寄与してしまうことが原因だと考えられる．このようなワークロードに関しては，逆に fairness を無視したブーストは効果があり，hackbench では約 9%向上している．

また，PLE の発生した回数を測定し，対策なしを元に正規化した．結果は図 6 である．全てのベンチマークについて，今回評価している手法の両方又はいずれかが有効であることがわかる．IPI の送信におけるバリア同期で PLE が発生しているものはノーマルモードの仮想 CPU をブーストする手法が有効で，スピンロックの待機待ちで PLE が発生しているものは fairness を無視したブーストが有効である．両方の手法で PLE の回数が減っているものは，両方の PLE の要因があると推測できる．

7. 関連研究

PLE 発生時の仮想 CPU スケジューリングについて，現在の KVM とは違う手法が提案されている．APPLES[13] は PLE の要因がスピンロックだという想定のもと，PLE を起こした仮想 CPU が素早くスピンロックを獲得可能となる仮想 CPU スケジューリングを提案している．このスケジューリング手法は IPI におけるバリア同期が要因の場合にも副次的に上手く機能する．そのため，著者の意図以上に大きなパフォーマンスの向上が表われていると考えられる．また，仮想 CPU 間の IPI のやり取りを考慮した仮想 CPU スケジューリングが提案されている [14]．これにより，IPI によるマルチスレッドアプリケーションの同期処理が効率的になり，パフォーマンスが向上する．

ゲスト OS の設計上，短時間で終わることを前提とした処理を実行している仮想 CPU がプリエンプトされることを防ぐための手法として，ゲスト OS と VMM の間で共有メモリを設けるなどし，セマンティックギャップを緩和する手法が提案されている [1] [6] [15] ．これにより VMM は短時間で終えたいタスクを実行している仮想 CPU を把握でき，それをプリエンプトすることを防ぐことができる．

また，仮想時間不連続性問題の中には過度に長いビジーウェイト以外のゲスト OS の意図していない挙動を引き起こすものもある．Blocked Waiter Wakeup 問題 [16] という仮想 CPU が頻繁に行われることで性能低下が起きる問題や，Read Copy Update(RCU) の同期処理において，その Reader がプリエンプトされることでメモリ使用量やパフォーマンスが低下する RCU-Reader Preemption 問題 [17] ，I/O 処理を行うタスクの実行が不連続になるこ

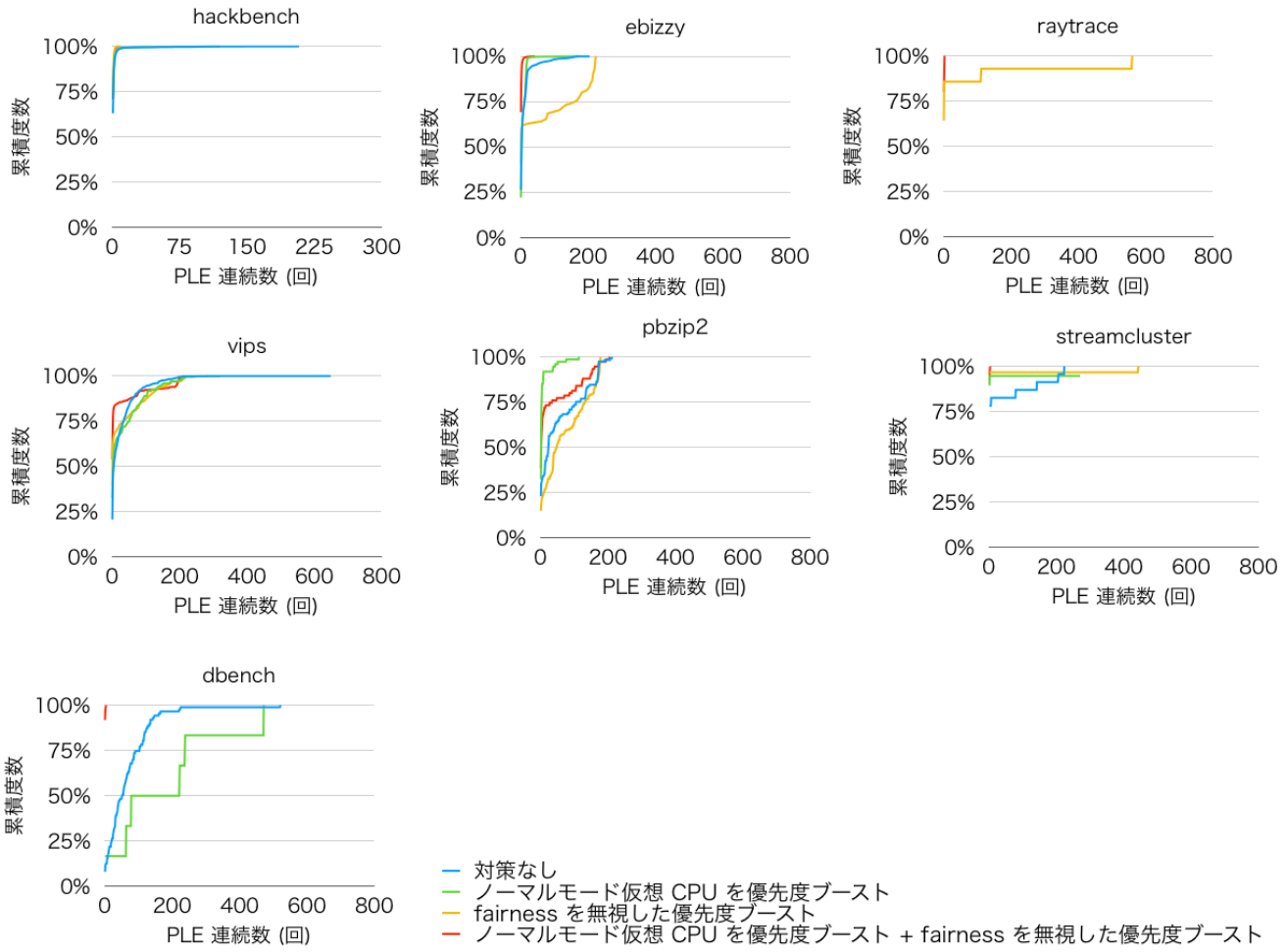

図 4　PLE の連続数の累積頻度グラフ

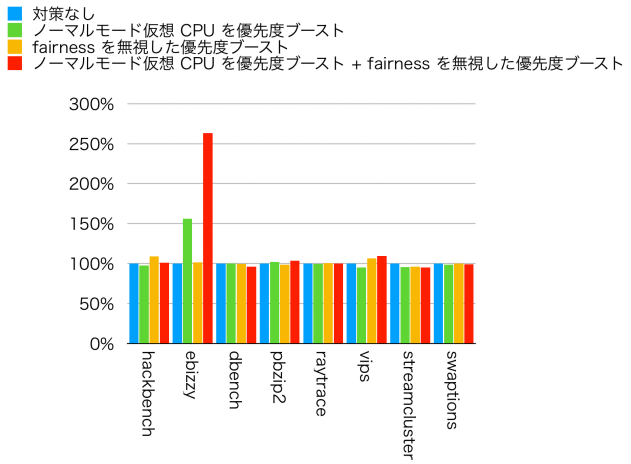

図 5　パフォーマンス

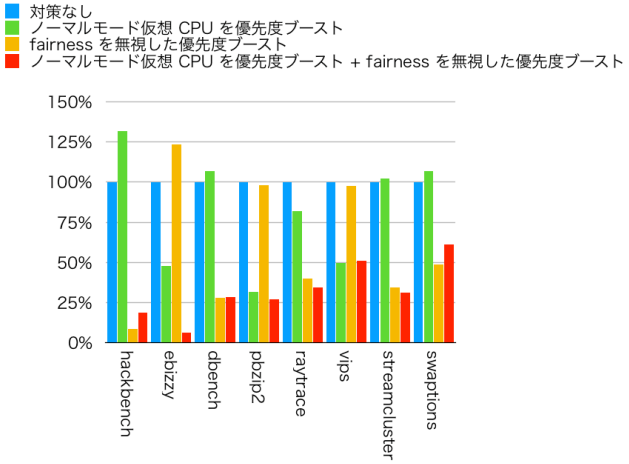

図 6　PLE 発生回数

とで I/O スケジューラが非効率な挙動をしてしまう I/O inactivity 問題 [18] などが報告されている．

8. まとめ

仮想化環境では，仮想時間の不連続性により，過度に長いビジーウェイトが起こることがある．PLE のようなハードウェア機能を用いてそれを検知することはできるが，VMM の仮想 CPU スケジューリングが不適切な場合，PLE が多発する．

現状の KVM の仮想 CPU スケジューリング手法において PLE の多発に繋がるケースを考察し，その対策手法を提案した．また，その内いくつかを実装して評価したとこ

ろ，パフォーマンスが最大 163%増加することを確認した．

9. 謝辞

本研究は，JST，CREST，JPMJCR19F3 の支援を受けたものである．

参考文献

[1] Ahn, J., Park, C. H., Heo, T. and Huh, J.: Accelerating Critical OS Services in Virtualized Systems with Flexible Micro-sliced Cores, *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA, ACM, pp. 29:1–29:14 (online), DOI: 10.1145/3190508.3190521 (2018).

[2] Dong, Y., Mallick, A., Nakajima, J. and Tian, K.: Extending Xen * with Intel Virtualization Technology (2006).

[3] : amd-v, `http://www.amd.com/en-us/solutions/servers/virtualization/`.

[4] Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A.: KVM: the Linux Virtual Machine Monitor, In Proceedings of the 2007 Ottawa Linux Symposium (OLS' -07 (2007).

[5] 直樹安野，健太石黒，健二河野：仮想環境における Pause Loop Exit の多発要因の調査，技術報告 15，慶應義塾大学, 慶應義塾大学, 慶應義塾大学 (2019).

[6] Teabe, B., Nitu, V., Tchana, A. and Hagimont, D.: The Lock Holder and the Lock Waiter Pre-emption Problems: Nip Them in the Bud Using Informed Spinlocks (I-Spinlock), *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, New York, NY, USA, ACM, pp. 286–297 (online), DOI: 10.1145/3064176.3064180 (2017).

[7] Raghavendra, K.: Virtual Cpu Scheduling Techniques for Kernel Based Virtual Machine (Kvm), pp. 1–6 (online), DOI: 10.1109/CCEM.2013.6684443 (2013).

[8] : Ebizzy, `https://sourceforge.net/projects/ebizzy/`.

[9] Bienia, C.: Benchmarking Modern Multiprocessors, PhD Thesis, Princeton University (2011).

[10] : Hackbench, `http://people.redhat.com/mingo/cfs-scheduler/tools/hackbench.c/`.

[11] : Dbench, `http://manpages.ubuntu.com/manpages/raring/man1/dbench.1.html`.

[12] : Pbzip2, `https://openbenchmarking.org/test/pts/compress-pbzip2`.

[13] Shan, J., Ding, X. and Gehani, N.: APPLES: Efficiently Handling Spin-lock Synchronization on Virtualized Platforms, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 28, No. 7, pp. 1811–1824 (online), DOI: 10.1109/TPDS.2016.2625249 (2017).

[14] Kim, H., Kim, S., Jeong, J., Lee, J. and Maeng, S.: Demand-based Coordinated Scheduling for SMP VMs, *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, New York, NY, USA, ACM, pp. 369–380 (online), DOI: 10.1145/2451116.2451156 (2013).

[15] Kashyap, S., Min, C. and Kim, T.: Scaling Guest OS Critical Sections with eCS, *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, Boston, MA, USENIX Association, pp. 159–172 (online), available from ⟨https://www.usenix.org/conference/atc18/presentation/kashyap⟩ (2018).

[16] Ding, X., Gibbons, P. B., Kozuch, M. A. and Shan, J.: Gleaner: Mitigating the Blocked-Waiter Wakeup Problem for Virtualized Multicore Applications, *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, Philadelphia, PA, USENIX Association, pp. 73–84 (online), available from ⟨https://www.usenix.org/conference/atc14/technical-sessions/presentation/ding⟩ (2014).

[17] Prasad, A., Gopinath, K. and McKenney, P. E.: The RCU-Reader Preemption Problem in VMs, *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, Santa Clara, CA, USENIX Association, pp. 265–270 (online), available from ⟨https://www.usenix.org/conference/atc17/technical-sessions/presentation/prasad⟩ (2017).

[18] Jia, W., Wang, C., Chen, X., Shan, J., Shang, X., Cui, H., Ding, X., Cheng, L., Lau, F. C. M., Wang, Y. and Wang, Y.: Effectively Mitigating I/O Inactivity in vCPU Scheduling, *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, Boston, MA, USENIX Association, pp. 267–280 (online), available from ⟨https://www.usenix.org/conference/atc18/presentation/jia⟩ (2018).