Vol.2016-OS-138 No.12
2016/8/8

Lock-Holder Preemption に対する Pause Loop Exiting の限界に関する調査

山崎 修平[1]　河野 健二[1]

概要：近年，仮想化環境は広く使用されるようになっている．複数のマシンを動かすためにはハードウェア等の物理資源を仮想化して使用する必要があり，ハイパーバイザ型の仮想化ではハイパーバイザが物理資源を管理する．この様な環境ではスピンロックを獲得することができない仮想 CPU (仮想 CPU) が長期間ビジーウェイトしたままになってしまう Lock-Holder Preemption (LHP) という問題が発生することが知られている．この様な問題に対して，いくつかのハードウェアでは一定時間以上のビジーウェイトした場合にはハイパーバイザに制御を移すような機能が追加されている．Intel ではこれを Pause Loop Exiting (PLE) という．しかし，PLE を利用するだけでは LHP の完全な対策になっておらず，PLE が頻発することによってオーバーヘッドが掛かることがある．本論文では PLE の発生状況と，それに対するスピンロックの獲得状況を分析し，より効率よく PLE を使用するための新しい仮想 CPU スケジューリングの方法について提案する．

キーワード：仮想化，Lock-Holder Preemption，Pause Loop Exiting

1. はじめに

仮想化環境はクラウドサービス等において広く利用されており，Google [1] におけるクラウドサービスや Amazon EC2 [2] なども仮想化技術を基盤としている．仮想環境上では CPU やメモリ等の物理資源を仮想化することによって，1 台の物理マシン上で複数の仮想マシン (VM) を動作させることができる．これにより，複数の仮想マシンをひとつの物理マシン上に集約することができ，計算資源の効率的な利用が可能となっている．特に，クラウド環境では CPU 利用率が平均的に低いことが知られており，集約によって CPU に利用効率を高める効果は大きい．実際，仮想化環境では物理 CPU は仮想 CPU として仮想化されており，物理 CPU の個数よりも多くの仮想 CPU を利用することが可能となっている．ハイパーバイザが仮想 CPU のスケジューリングを行うことによって，物理 CPU の個数を超えた仮想 CPU が利用可能となっている．

しかし，仮想マシン内で動作するゲストオペレーティングシステムは，常に物理 CPU を占有しているという前提で設計されているため，仮想 CPU のスケジューリングによってゲストオペレーティングシステム内の制御に問題が生じる場合がある．Lock-Holder Preemption (LHP) はこうした問題のひとつである．LHP は次のような状況において生じる．スピンロックを保持している仮想 CPU がスピンロックを解放する前に，仮想 CPU の切り替えが起きると，他の仮想 CPU はそのスピンロックを獲得することができず長く待たされてしまう．その結果，スピンロックを保持している仮想 CPU がスケジュールされるまで，同じスピンロックを獲得しようとする仮想 CPU は CPU 時間を浪費することになってしまう．

LHP による問題を緩和するため，Intel 社 [3] や AMD 社 [4] では，仮想化支援のためのハードウェア機能のひとつとして，Pause Loop Exiting (PLE) あるいは Pause Filtering (PF) という機能を提供している．PLE は Intel 社における呼称であり，PF は AMD 社における呼称である．以降，両者を特に区別せず PLE と呼ぶ．PLE では，一定時間以上スピンロックの獲得が行われない場合，VMExit により仮想マシンからハイパーバイザに制御を移す仕組みである．スピンロックの実装においては，ビジーウェイトを行うループ中で PAUSE 命令を実行することが推奨されており，通常のオペレーティングシステムであれば必ずそのように実装されている．PLE では一定の頻度以上で PAUSE 命令が連続する場合，VMExit を行う仕組みである．

PLE が実装される以前から LHP を緩和する手法が提案

[1] 慶應義塾大学
Keio University

されている．Uhlig ら [5] は，ゲストオペレーティングシステムの時間を安全に切り替えられる時間と，スピンロックを保持している可能性がある時間の 2 種類に分類した．ゲストオペレーティングシステム自体には手を加えることがない方法を取っており，ユーザモードでゲスト VM が動いている時間と確実にスピンロックを保持していない時間にのみプリエンプションを許可している．スピンロックを保持している可能性がある時間を多めに見積もっているため，必要以上に仮想 CPU に時間を割り当てることで VM 間の公平性を欠いてしまうという可能性がある．また，仮想 CPU の優先度を変更することによってスピンロックを確保している時には，ハイパーバイザによる仮想 CPU の切り替えが起こらないようにする研究 [6] もある．

PLE を利用した場合であっても，LHP が発生した際にハイパーバイザに制御が移されるに過ぎないため，LHP の根本的な解決策とはなっていない．実際，Intel 社の PLE を使用し，ハイパーバイザとして Xen を用いた実験を行ったところ，実行のタイミングに応じて PLE が頻発するケースがあり，PLE の発生頻度と実行時間の間に正の相関があることがわかった．実験ではベンチマークの実行時間は 2 秒に満たないにも関わらず，25 万回以上の PLE が発生するケースが観測されている．この時，通常の実行時間は約 1.5 秒 なのに対し，PLE が頻発しているケースでは 1.8 秒以上時間が掛かる．PLE 発生時にスピンロックを保持する仮想 CPU が適切にスケジューリングされていれば，特定のケースにおいて PLE が頻発することは考えにくく，PLE 発生後もスピンロック獲得待ちの仮想 CPU が何度もスケジューリングされているのではないかと考えられる．

この結果を受けて，本論文では PLE 発生時の状況をより詳しく分析するための実験を行う．ゲストオペレーティングシステム上でスピンロックの獲得状況を記録し，ハイパーバイザ上で PLE の発生状況を記録する．両者の照合し，スピンロックの獲得状況に対する PLE の発生状況を観測する．その結果，次の 3 点が明らかになった．

- 複数の仮想 CPU が同時期に連続して PLE を引き起こすことがある．これは，PLE の再発を防ぐように仮想 CPU スケジューリングが適切に行われていないことを示している．これは，スピンロック獲得のタイミングによって PLE が頻出し，実行時間が延びる一因となっている．
- ある仮想 CPU がスピンロックを獲得しているために，他の仮想 CPU がスピンロックを獲得できずに PLE が発生するケースがある．これは，LHP の問題が解決されていないことを示している．
- スピンロックを獲得している仮想 CPU が存在しない場合でも PLE が頻出するケースがある．これは，ゲストオペレーティングシステムとして採用した Linux では，スピンロックの実装にチケット・スピンロックを使用していることが原因だと考えられる．チケット・スピンロックはスレッド間の公平性を保障するための仕組みであり，スピンロックの獲得要求を出した順にチケットが与えられ，その順序に従ってスピンロックが獲得できる仕組みである．チケット・スピンロックを用いた場合，スピンロックが解放されていても，チケットの順番通りに仮想 CPU をスケジューリングしなければスピンロックは獲得できない．これにより PLE が頻発していると考えられる．

これらの問題を改善するために，PLE 発生時の仮想 CPU のスケジューリング方法の改善を提案する．PLE 発生時の仮想 CPU の割り当てを変更することによって，PLE の発生回数を抑えることを目的とする．PLE が連続で発生することを防ぐことができれば，ゲストのオペレーティングシステムを改変する事無く LHP の影響を緩和することができる．

本論文の構成を以下に示す．2 章では仮想化環境で発生する代表的な問題について述べる．3 章では PLE の発生回数と，実行時間に関する調査について述べる．4 章では実際の PLE の発生状況を詳細に分析した結果と，対応策のスケジューリング方法について述べる．5 章では関連研究について説明する．6 章では本論文のまとめを述べる．

2. 仮想化環境におけるスピンロックの問題

2.1 Lock-Holder Preemption (LHP)

仮想化環境では 1 つの物理 CPU (物理 CPU) を複数の仮想 CPU に割り当てながら使用する．仮想化の方式にはホストオペレーティングシステム型とハイパーバイザ型とがあるが，ハイパーバイザ型の仮想化ではどの 物理 CPU をどの仮想 CPU に割り当てるかはハイパーバイザによって決定される．ハイパーバイザによって割り当てられた時間が消費されると，ゲストオペレーティングシステムの状況に関わらず仮想 CPU の切り替えが行われる．問題となるのはゲストオペレーティングシステム上でスピンロックが確保されていた場合，そのスピンロックが解放されないまま仮想 CPU が切り替えられることがある点である．スピンロックを保持したまま解放されない仮想 CPU ができてしまい，他の仮想 CPU からは同じスピンロックを獲得することができない状態になる．

Linux カーネルではスピンロックはすぐに解放されていることを前提にしているため，スピンロックを獲得できない時はビジーウェイトする設計になっている．しかし，スピンロックを保持している仮想 CPU が切り替えられている状況では，再びその仮想 CPU に CPU 時間が割り当てられるまでスピンロックの解放が行われない．そのため，同じスピンロックを獲得しようとする仮想 CPU は割り当てられた CPU 時間をビジーウェイトしたまま消費する．この問題を Lock-Holder Preemption (LHP) という [5].

LHP によって仮想 CPU がビジーウェイトしている時，その仮想 CPU はプリエンプションが禁止されているため他のスレッドを動かすこともできない．通常のスピンロックはその殆どが 20 μsec 以下の時間しか確保されないが，CPU の切り替えが行われてしまうと数十 msec 以上確保されていることがある．そのため，スピンロックを保持している仮想 CPU は動いていないにも関わらず，数十 msec 以上スピンロックの獲得待ちに時間を使ってしまう可能性がある．

2.2 Pause Loop Exiting(PLE)

LHP を回避するための機能として，Intel では Pause Loop Exiting (PLE) という機能が提供されている．これは PAUSE 命令の連続を検知して，ゲストオペレーティングシステムからホストオペレーティングシステムに制御を戻すことができる機能である．PLE を使用するには PLE_Gap と PLE_Window という 2 つの値をあらかじめ設定しておく．PLE_Gap は PAUSE 命令が連続していると認識するまでの最大時間であり，PLE_Window は PAUSE 命令が連続して実行される最大時間を示している．PLE_Gap 以内の間隔で PAUSE 命令が実行されると PAUSE 命令が連続していると認識され，PLE_Window 以上の時間 PAUSE 命令が連続すると PLE が発生する．

PLE_Gap は PAUSE 命令が連続した時に認識するかどうかの差になるので，閾値前後で PLE が発生するか全く発生しなくなるかが変わる．例えば PLE_Gap を 32 cycles に変更した場合，PLE は全く発生しなくなった．一方で，PLE_Window は値によって PLE の発生回数に影響を与える．PLE_Window の値が極端に短い場合には，スピンロックを保持している仮想 CPU が正しく動いているにも関わらず，LHP が発生していると誤認して PLE を発生させてしまう可能性がある．しかし，LHP の検出自体は非常に早くすることができる．反対に PLE_Window が長い場合，LHP を誤認することはなくなるが LHP を検出するまでの時間が長くなってしまう．極端な場合には PLE 自体が発生しなくなる可能性がある．KVM や Xen ではこれらの影響が考慮した数値がそれぞれ設定されおり，Xen-4.5.0 のデフォルトでは PLE_Gap は 128 cycles，PLE_Window は 4096 cycles に設定されている．

3. PLE の発生回数と実行時間

3.1 仮想 CPU の数の増加

近年，1 つのハードウェア上に 40–60 の VM が動くということは当たり前になっている [7]．これは，16 コアのマシンでは 2–4 個の VM が 1 つの 物理 CPU 上で動くということを意味している [8]．物理 CPU に割り当てられる仮想 CPU の数が多いほど仮想 CPU の切り替えが行われやすくなり，仮想 CPU の切り替えが行われやすい環境

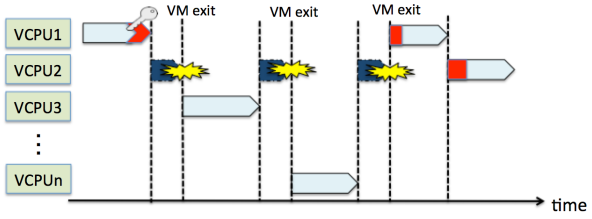

図 1 仮想 CPU の増加に対する LHP の問題点

のほど LHP は発生しやすくなる．今後 CPU の性能が上がるにつれて 1 つの物理マシンで動かすことのできる VM は多くなることが予想されるが，この問題はより深刻化する可能性がある．

また，1 つの VM に複数の仮想 CPU が割り当てられている場合，LHP を解消するのが難しくなる場合がある．例えば，上の図 1 では 仮想 CPU1 がスピンロックを保持した状態で 仮想 CPU2 が同じスピンロックを獲得しようとしている状況を表している．仮想 CPU2 は PLE を引き起こすが，仮想 CPU2 が動き出すためには 仮想 CPU1 がスピンロックを解放しなければならない．ハイパーバイザからはどの仮想 CPU がスピンロックを保持しているかを知ることはできないため，スケジューリングアルゴリズムに従って仮想 CPU を動かすことになる．図 1 の様に PLE 発生時に 仮想 CPU3 を動かしてから仮想 CPU2 を動かすと，スピンロックはまだ解放されていないため 仮想 CPU2 は再び PLE を引き起こす．VM あたりの仮想 CPU 数が増加するにつれてスピンロックを保持する仮想 CPU を推定するのは難しくなり，1 度の LHP で複数回の PLE が発生する確率が高くなる．

3.2 PLE の多発による遅延

PLE が発生するとゲストオペレーティングシステムとホストオペレーティングシステムの切り替えを行わなければならないため，何度も PLE が頻発するようなことになると少なからずオーバーヘッドが掛かる可能性が考えられる．そこで，PLE による遅延発生の可能性を確かめるために Xen 環境下で同じベンチマークを 10,000 回実行する実験を行った．PLE の設定はデフォルトのままで，4 つの物理 CPU に対して 32 の仮想 CPU を割り当てている．

図 2 では，その時のベンチマークの実行時間と PLE の発生回数の関係を表している．ほとんどの場合では約 1.5 sec の実行時間になっている．しかし PLE が多発するケースが存在し，それに伴って遅延が発生していることが分かる．多い時では 2 sec 以下の実行時間に対して，約 25 万回の PLE が発生している．遅延が発生しているときに必ずしも PLE が多発しているわけではないが，PLE が多発している時には少なからず遅延が発生しているため，両者の間には相関があることが伺える．

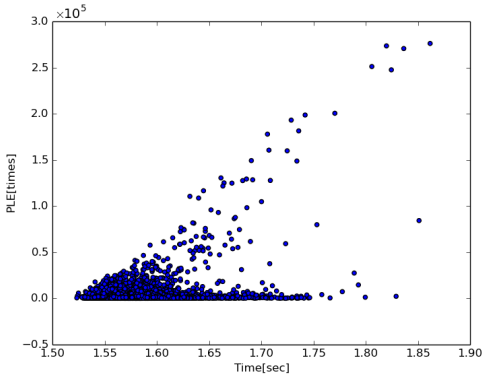

図 2 PLE の発生回数と遅延

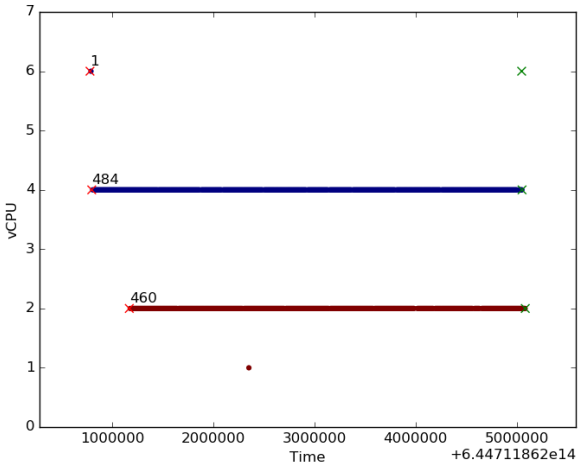

図 3 スピンロックの獲得までに発生する PLE

4. PLE 発生状況の調査

PLE の頻発によって遅延が発生している状況は確かめることができたが，実際 1 回の LHP に対して PLE が複数回発生している状況が担っているのかは不明のままである．そこで，PLE が頻発している状況をより詳細に確かめるための実験を行った．ゲストオペレーティングシステム側ではスピンロックの獲得状況を記録し，ホストオペレーティングシステム側では PLE の発生状況を記録した．

4.1 調査環境

実験には Intel Xeon 2.67GHz 6 コア，メモリ 4GB のマシンを使用した．また，仮想化ソフトウェアの Xen-4.5.0 を使用し，ホストのオペレーティングシステムには Ubuntu 15.04 をインストールした．ゲスト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，カーネルのバージョンは Linux kernel 3.19 をビルドした．CPU の割り当てはホストオペレーティングシステムに 2 物理 CPU, ゲストオペレーティングシステムに 4 物理 CPU をそれぞれ割り当て，ゲストオペレーティングシステムは 8 仮想 CPU で動作させた．ゲストオペレーティングシステムで動作させるベンチマークには Sysbench の OLTP を使用し，MySQL 内の 100,000 件のデータベースの中からランダムに 10,000 件を検索するプログラムを 128 スレッド並列で行った．このベンチマークを実行させることで，PLE を発生させながら次の記録を行った．

4.2 PLE の連続性

セクション 3.2 で述べたように PLE が頻発する時がある．そこで，PLE の発生状況をより詳しく確かめる他の実験を行った．ホストオペレーティングシステムでは PLE 発生時に PLE を起こした仮想 CPU の番号，PLE した時点のタイムスタンプカウンタ (TSC)，プログラムカウンタ (PC) の値をそれぞれ記録した．ゲストオペレーティングシステムではスピンロックを獲得しようとした時点を try_time とし，実際に獲得することができた時間 acq_time との時間の差を計測した．

try_time と acq_time は通常であればそこまで長い時間にならないため，スピンロックを獲得できたときには非常に短い時間で獲得できることになる．しかし，LHP が発生しているときはスピンロックを長時間獲得できないため，この try_time と acq_time の間隔が非常に長くなると考えられる．とこで，try_time と acq_time の間に一定時間以上の時間差があった場合のみを記録し，その時の TSC の値と仮想 CPU の番号などを記録した．

図 3 はあるスピンロックの獲得に対して PLE が連続で発生している状況を表している．図は縦軸が仮想 CPU の番号であり，横軸が TSC をクロック単位で表示している．つまり，図の左から右へと時間経過が表されていて，各仮想 CPU がそれぞれいつ PLE を引き起こしたのかとスピンロックの獲得を行ったのかを記録している．図 3 内で赤い印が付いている場所は try_time, 緑の印が付いている場所は acq_time をそれぞれ示している．また，青と茶色の線のように見える部分は PLE 発生時を記録した丸が連なっている．

try_time の上につけられている数字は，そのスピンロックの try_time から acq_time までに発生している PLE の回数である．図 3 では 仮想 CPU4 で 484 回， 仮想 CPU2 で 460 回の PLE が発生していることが分かる．つまり，それぞれの仮想 CPU は 1 つのスピンロックを獲得するまでに， 450 回以上も PLE を繰り返すことによって CPU 時間を消費していることが分かる．

4.3 PLE 連続発生の原因

PLE が連続で発生しているということは，その間スピンロックの解放が行われていない可能性が高い．そこで PLE が連続で発生する原因を突き止めるために，try_time

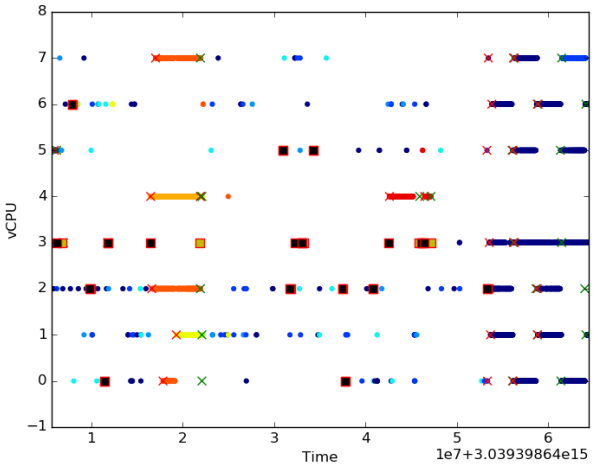

図 4 スピンロックの獲得状況 PLE の発生状況

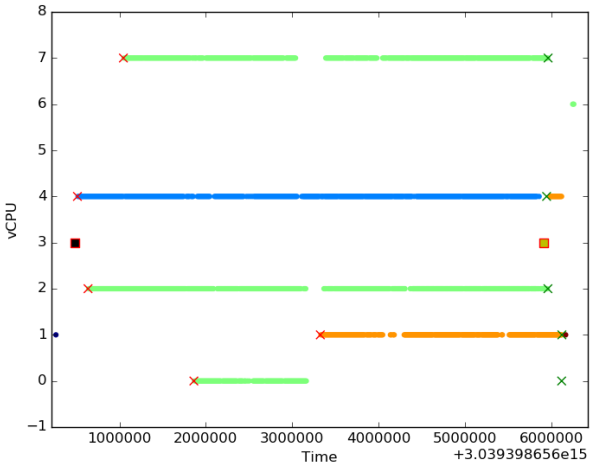

図 5 スピンロックの解放が行われないことによる PLE の連続

と acq_time に加えてスピンロックを解放した時間である rls_time を記録するようにゲストオペレーティングシステムに改変を加えた．スピンロックの獲得から解放までに関してはカーネルビルドにおけるスピンロックの獲得時間の 99.5% が 2048 cycles 以下であるというデータがある．また，PLE_Window の設定は 4096 cycles になっていることからも，これ以上の時間スピンロックを確保しているということは通常では起こらないはずである．そこで，スピンロックを確保してから解放するまでの時間である rls_time − acq_time が 30,000 cycles 以上になった時のみを記録し，その時の TSC と仮想 CPU 番号を記録した．

図 4 はその時のスピンロックの獲得状況と PLE の発生状況を表している．図 3 と違うのは，赤で囲われた四角で表示されている部分である．四角で表示されたもののうち，内側が青くなっているのがスピンロックを獲得した時間であり，内側が黄色で表示されているのがスピンロックを解放した時刻になっている．

図 4 を見るとスピンロックの解放までに長い時間が掛かっている部分がいくつかあるのが分かる．次のセクションではスピンロックの保持に対する PLE 連続発生のパターンを 2 つに分けて説明する．

4.3.1 過剰なスピンロックの保持

図 5 は図 4 の左側の一部を拡大したものである．この図では仮想 CPU 0,1,2,4,7 でそれぞれ PLE が連続して発生している．この原因として，その直前に行われているスピンロックの獲得が考えられる．仮想 CPU 3 が青い四角のところでスピンロックを保持したまま，黄色い四角の部分までスピンロックの解放が行われていない様子がわかる．仮想 CPU 3 が切り替えられてしまうことによって，長時間スピンロックを獲得することができずに PLE が連続で発生していることが確かめられた．

この図 5 の問題点は，PLE 発生時の仮想 CPU の割り当て方にある．PLE が発生した時スピンロックを保持している仮想 CPU が動いていない可能性を考えると，PLE した仮想 CPU を再び割り当てても PLE が発生するだけである．これでは PLE を設定していない状態とおなじになってしまうので，PLE した仮想 CPU は他の仮想 CPU を動か方が効率よく仮想 CPU を使うことできる．

4.3.2 Lock Waiter Preemption (LWP)

図 6 は図 4 の右側の一部を拡大したものである．スピンロック解放の黄色い四角は獲得の青い四角に重なっているため分かりにくいが，セクション の時とは違いスピンロックを保持したままの仮想 CPU は無い様子が分かる．しかし，はじめにスピンロックを一定時間保持した仮想 CPU に対して PLE が発生し，その後スピンロックは解放されているにも関わらず PLE が発生している．

この原因として考えられるのは Lock Waiter Preemption (LWP) という問題である [9]．LWP というのは Linux でスピンロックの実装にチケット・スピンロックが使用されているために発生する．チケット・スピンロックではスピンロック獲得待ちの CPU に対して公平性がより高く保たれるように，スピンロックを獲得しようとした CPU と同じ順番でしかスピンロックを獲得できないようになっている．

例えば，最初にスピンロックを獲得しようとした CPU がチケット 1 を持っているとすると，次に同じスピンロックを獲得しようとする仮想 CPU はチケット 2 を持つことになる．この時，チケットの数字通りの順番でしかスピンロックの獲得は行われない．こうすることで，スピンロックを獲得しようとする CPU が複数あるとき時に，タイミングの問題で長時間スピンロックを獲得することができない CPU が生まれないようにする．

しかし，仮想環境上ではチケットの順番通りにしか獲得できないことによる遅延が発生する．チケット 1 でスピ

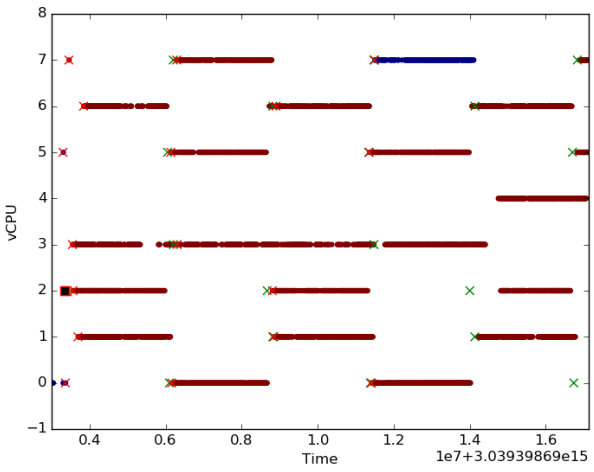

図 6 チケットの順番待ちによる PLE の連続

ンロックの獲得待ちをしている仮想 CPU がいる場合，その仮想 CPU が切り替えられているとチケット 2 以上の仮想 CPU からはそのスピンロックを獲得することができない．つまり，獲得したいスピンロックが解放されているにも関わらず，その仮想 CPU よりも前のチケットを持つ仮想 CPU が切り替えられている状態ではスピンロックの獲得は不可能になる．この状況を LWP という．

図 6 では最初にスピンロックの一定時間保持されており，解放はすぐにしているが PLE が連続している．最初にスピンロックを獲得できずにチケットが小さい数字の仮想 CPU が切り替えられており，LWP による PLE が連続して発生している可能性がある．

4.4 ハイパーバイザによる改善案

セクション 4.3.2 とセクション 4.3.2 で異なる点はスピンロックを保持している仮想 CPU があるのか無いのかという点にある．どの仮想 CPU がスピンロックを持っているのかが重要な場合と，スピンロックを保持している仮想 CPU 自体が存在していない場合に分かれている．こうした点から，ハイパーバイザ側からどの仮想 CPU を順番に動かせば効率よく仮想 CPU を動かすことができるのかを推測するのは難しい．

しかし，どちらの問題時にも PLE が同じ仮想 CPU で一定時間連続しているという点では共通している．PLE 発生時に同じ仮想 CPU を動かすということは，PLE を使用しないで CPU 時間を浪費している状態と同じである．そのため，PLE によってハイパーバイザに制御が移った時，同じ仮想 CPU にタイムスライスを割り当てないようにスケジューリングする．これにより，より短い時間でスピンロックを獲得することが可能になるはずである．

5. 関連研究

仮想化環境のおける並列処理の遅延は古くから問題とされており [10]，これを改善するために多くの研究が行われている．LHP という問題を提起した研究では，ゲストオペレーティングシステムの動いている時間をプリエンプションが可能な safe state と unsafe state に分けることによって，スピンロックを保持している仮想 CPU をプリエンプションしないようにする手法を提案している [5]．しかし，この方法ではゲストオペレーティングシステムを切り替える時間が限られてしまうため，適切なタイミングで仮想 CPU の切り替えが行われない可能性がある．

こうした問題が広く取り扱われるのに従って，ハードウェアによる支援が追加された [3], [4]．これらをハイパーバイザのスケジューラを対応させることで，LHP を緩和することが可能となった [11]．また，ゲストやホストのスケジューラを変更することによって，LHP 等の遅延を避ける手法も提案されている [6], [12], [13], [14]．Gleaner[15] は Blocked Waiter Wakeup Problem (BWW) という新たな問題に焦点を当て，できるだけ仮想 CPU の切り替えが起こらないようにタスクを統合することによって，ブロック同期における仮想環境上での遅延問題の改善を提案している．

vScale[8] では仮想 CPU を動的に変更することで仮想環境の遅延を緩和できるという研究 [16] を元に，Hot Plug では不可能だった高速な仮想 CPU の動的変更を実現している．仮想 CPU の数を適切に保ちながら VM を動かすことにより，LHP に限らない仮想環境での並列処理に関する問題を緩和することができる．

また，LWP に対してプリエンプション可能なチケット・スピンロックの実装を行う研究もある [9]．Linux の実装に直接手を加えてチケットの順番に限らないスピンロックの獲得を可能にすることによって，LWP による影響を減らしている．

一部の研究ではゲストに手を加えないまま LHP を改善しているが，ゲストの仮想 CPU がどのタイミングでスピンロックを確保しているかを特定することが難しいため，必要以上に CPU 時間を与えてしまう可能性がある．スケジューリングによる手法ではより効率よく仮想 CPU を割り当てることができるが，いずれの方法でもゲストオペレーティングシステム自体に手を加えることを前提としている．本論文ではハイパーバイザのスケジューラのみによる PLE の遅延を緩和することによって，PLE をより効率的に運用して汎用性の高い手法を提案する．

6. まとめ

仮想環境で問題として知られている LHP に対して，PLE

や PF といったハードウェアによる支援を用いることである程度問題を緩和することが可能である．しかし，現状 PLE を使用するだけでは PLE 自体が頻発するケースがあり，LHP による影響が少なくないことが分かった．また，PLE が発生する状況をゲストオペレーティングシステムのスピンロックの獲得状況と照らし合わせることによって，1 つのスピンロックの獲得に対して複数回 PLE が発生していることを確かめた．さらに，Linux カーネルではチケット・スピンロックが使用されていることによって，スピンロックを順番通りにしか獲得することができず，LHP 発生時に PLE の回数を冗長させてしまう可能性があることが分かった．今後はハイパーバイザのスケジューラに手を加えることによって，1 回の LHP 発生時に怒る PLE の回数を減らすようなスケジューラアルゴリズムを設計する．結果としてゲストオペレーティングシステムを改変すること無く，完全仮想化方式を維持したまま LHP による影響が小さくなるスケジューラを実装する．

参考文献

[1] : google. `http://www.google.com/`.
[2] : Amazon. `https://aws.amazon.com/jp/ec2/`.
[3] Dong, Y., Li, S., Mallick, A., Nakajima, J., Tian, K., Xu, X., Yang, F., Yu, W. and Yaozu DongShaofan LiAsit MallickJun NakajimaKun TianXuefei XuFred YangWilfred Yu: Extending Xen with Intel Virtualization Technology., *Intel Technology Journal*, Vol. 10, No. 3, p. 193 (2006).
[4] : amd-v. `http://www.amd.com/en-us/solutions/servers/virtualization/`.
[5] uhlig, v., levasseur, j., skoglund, e. and dannowski, u.: towards scalable multiprocessor virtual machines, *proceedings of the 3rd virtual machine research and technology symposium*, pp. 43 – 56 (2004).
[6] jiang, w., zhou, y., cui, y., feng, w., chen, y., shi, y. and wu, q.: cfs optimizations to kvm threads on multi-core environment, *proceedings of the international conference on parallel and distributed systems - icpads*, pp. 348–354 (2009).
[7] Pfaff, B., Pettit, J., Koponen, T., Amidon, K., Casado, M. and Shenker, S.: Extending networking into the virtualization layer, *In: 8th ACM Workshop on Hot Topics inNetworks (HotNets-VIII).New YorkCity,NY(October 2009)*.
[8] cheng, l. and lau, f. c. m.: vscale: automatic and efficient processor scaling for smp virtual machines, *eurosys*, Vol. 1, No. c (2016).
[9] ouyang, j. and lange, j. r.: preemptable ticket spinlocks : improving consolidated performance in the cloud, *vee*, pp. 191–200 (2013).
[10] karlin, a. r., li, k., manasse, m. s. and owicki, s.: empirical studies of competitve spinning for a shared-memory multiprocessor, *acm sigops operating systems review*, Vol. 25, No. 5, pp. 41–55 (1991).
[11] Dong, Y., Zheng, X., Zhang, X., Dai, J., Li, J., Li, X., Zhai, G. and Guan, H.: improving virtualization performance and scalability with advanced hardware accelerations, *IEEE International Symposium on Workload Characterization, IISWC'10* (2010).
[12] weng, c., liu, q., yu, l. and li, m.: dynamic adaptive scheduling for virtual machines, *. . . of the 20th international symposium on high . . .*, pp. 239–250 (2011).
[13] sukwong, o. and kim, h. s.: is co-scheduling too expensive for smp vms?, *eurosys'11 - proceedings of the eurosys 2011 conference*, pp. 257–271 (2011).
[14] kim, h., kim, s., jeong, j., lee, j. and maeng, s.: demand-based coordinated scheduling for smp vms, *asplos*, p. 369 (2013).
[15] 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)*, pp. 73–84 (2014).
[16] song, x., shi, j., chen, h. and zang, b.: schedule processes, not vcpus, *proceedings of the 4th asia-pacific workshop on systems - apsys '13*, pp. 1–7 (2013).