

仮想環境における Pause Loop Exit の多発要因の調査

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。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ため、仮想 CPU のプリエンプトにより仮想時間の不連続性が発生する。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。

現状の仮想化環境では PLE が多発することがあることが知られており、本研究では KVM を対象にその要因を定量的に分析する。その結果、1) プロセッサ間割込み (IPI) 処理におけるバリア同期、2) スピンロックの獲得待ち、3) タイムスタンプベースの待機処理、という 3 つのケースで PLE が多発していることを示す。さらに、KVM では PLE の多発を避けるように仮想 CPU スケジューリングを行っているものの、現状では十分に機能していないことを示す。

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 をスケジューリングするのが望ましい。PLE と 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

¹ 慶應義塾大学
Keio University

適切に行われないと、PLE が多発する。現状の VMM でも PLE の多発を防ぐための処理がなされているにも関わらず、KVM を対象とした我々の調査では、PLE が 27 回以上連続して発生するケースが全体の 50%以上を占め、最悪ケースでは 268 回 PLE が連続しているケースも確認できた。このことから、VMM の仮想 CPU スケジューリングが適切に行われていないと言える。

本稿では、PLE の発生要因の定量的な調査を行う。これにより、PLE 発生時の仮想 CPU スケジューラにおける PLE の多発を避けるために必要な要素を整理する。また、KVM の仮想 CPU スケジューラのログを取り、現状の PLE の多発を避ける仕組みが十分に機能していない原因を調査する。その結果、以下の 2 点が明らかになった。

- 主に 3 つの要因で PLE が発生している。
 - (1) プロセッサ間割込み (IPI) の送り先の仮想 CPU がプリエンプトされていることを起因とする、送り元の仮想 CPU の長時間のバリア同期
 - (2) スピンロックを保持した仮想 CPU がプリエンプトされていることを起因とする、ロックを獲得しようとする仮想 CPU の長時間の獲得待ち
 - (3) タイムスタンプベースの待機処理によるビジーウェイト
- KVM の PLE の多発を避けるためのしくみにより優先度がブーストされた仮想 CPU は、実際には優先的にスケジューリングされず、これが原因で PLE が多発している。

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。4 章では調査環境及びその結果を示す。5 章では関連研究について述べる。6 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。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 の停止により、意図していない挙動が起きることがある。

そのような挙動の一つとして、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、短期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 [4] 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。

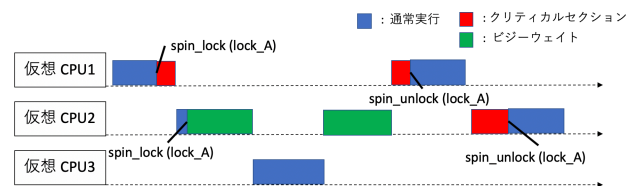


図 1 LHP 発生の様子

2.2 Pause Loop Exit(PLE)

このような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、他に仮想 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。

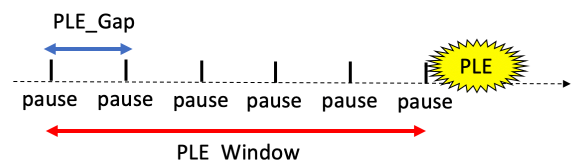


図 2 PLE の起こる様子

3. PLE の多発

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。

現状の KVM は、PLE 発生時に Candidate VCPU selection[5] 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。

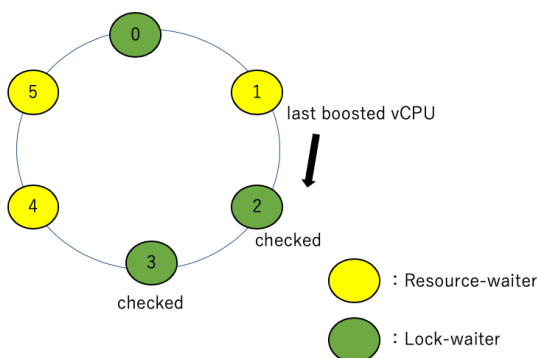


図3 PLE 時の KVM の仮想 CPU スケジューリング

しかしながら、KVM にはこのように 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

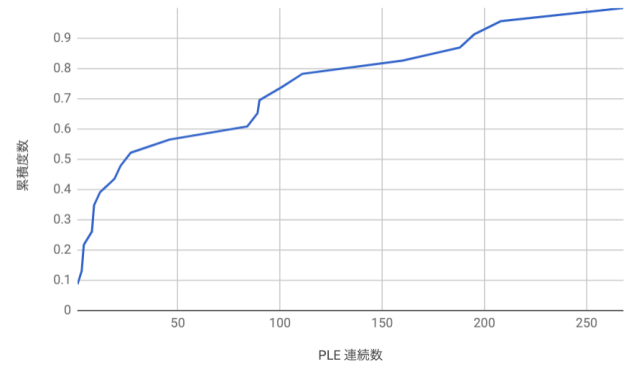


図4 PLE の連続発生回数

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。実験環境の詳細は4章の通りである。50%以上の割合で PLE が27回以上、最大で268回連続的に発生しており、PLE が多発していると言える。

4. PLE の多発要因の調査

4.1 調査方法

PLE 発生時、仮想 CPU のインストラクションポイント (IP) を記録するトレースイベントを新たに追加した。これにより得られたアドレスを、ゲスト OS を逆アセンブリして得られたバイナリのアドレスと関数名の対応から、PLE が発生した関数を調べた。得られた関数の性質に応じて、PLE の要因を分類分けした。

また、KVM の PLE 時の仮想 CPU スケジューリングが PLE の多発を避けるように上手く機能しているか調査した。PLE 発生時に、KVM が優先度をブーストした仮想 CPU を記録するトレースイベントを新たに追加し、既存のトレースポイントとあわせてログを収集した。

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 18.04 LTS 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 14.04 LTS 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、固定値を用いた。

ベンチマークはマルチスレッドベンチマークである ebizzy[6], PARSEC[7], hackbench[8], dbench[9], pbzip2 [10] を使用した。1つの仮想マシン上で PARSEC の swap- tions 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。

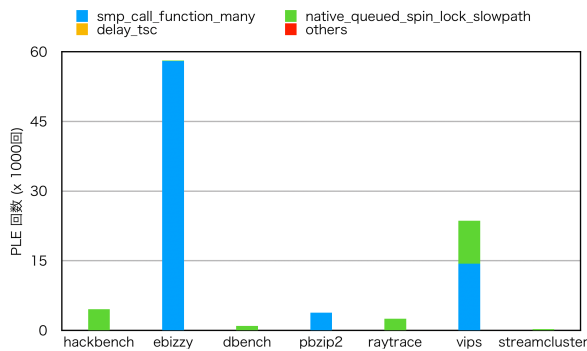


図 5 PLE が発生した関数 (PLE_Window 32768)

4.2 PLE の発生要因

PLE の発生要因の実験結果が、図 5 である。結果より、ベンチマークごとにその割合は違うが、全体として主に 3 つの関数で PLE が起こっていることがわかった。

1 つ目は `smp_call_function_many` 関数で、これはプロセッサ間割り込み (IPI) を複数のコアに向けて送るための関数である。IPI 処理において、IPI の送り元が送り先コアが処理を終えるのを待つバリア同期でビジーウェイトすることがある。図 6 は `smp_call_function_many` 関数の擬似コードである。IPI を送り先が処理するまでバリア同期する時、8 行目から 12 行目が実行される。12 行目の `wait` 関数は、それぞれの CPU が IPI を処理し終わったことを確認できるまでビジーウェイトする関数である。`smp_call_function_many` 関数で PLE が発生する時、このビジーウェイトが PLE を引き起こしていると言える。このケースでは IPI の送り先の仮想 CPU がブリエンプトされていることが要因でビジーウェイトが過度に長くなっていると考えられ、送り先の仮想 CPU を優先的にスケジューリングする必要がある。

2 つ目は `native_queued_spin_lock_slowpath` 関数で、スピンロックの獲得待ちをする関数である。この関数で起こった PLE は LHP が原因であり、スピンロックを保持した仮想 CPU を優先的にスケジューリングする必要がある。

3 つ目は `delay_tsc` で、タイムスタンプベースの待機処理を行う関数である。これは、一定時間ビジーウェイトすること自体を目的とした関数である。このケースにおいては一定時間必ずビジーウェイトをする。

4.3 PLE_Window と PLE の要因の関係

PLE_Window の値に応じた PLE の発生要因の傾向の変化を調査した。図 7 から図 10 がその結果である。全てのケースにおいて、`smp_call_function_many` 関数、`native_queued_spin_lock_slowpath` 関数、`delay_tsc` 関数の 3 つ以外で発生した PLE は 1%以内であり、PLE_Window の関わらず、前述の 3 つが主要な要因となっていることが

```

1 function smp_call_function_many(const struct
2   cpumask func *mask, smp_call_func_t func,
3   void *info, bool wait){
4
5   /* Omitted */
6
7   arch_send_call_function_ipi_mask(cfd->
8     cpumask);
9
10  if (wait) {
11    for_each_cpu(cpu, cfd->cpumask) {
12      struct call_single_data *csd;
13
14      csd = per_cpu_ptr(cfd->csd, cpu);
15      csd_lock_wait(csd);
16    }
17  }
18 }

```

図 6 `smp_call_function_many` 関数のコード

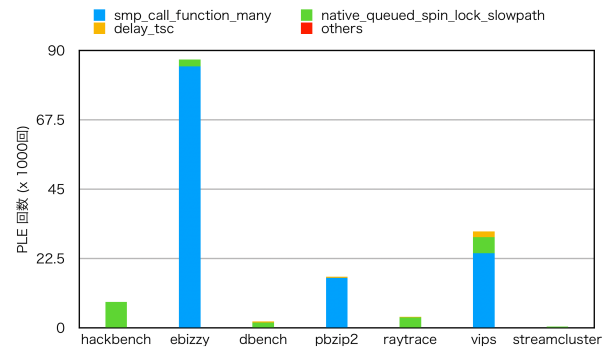


図 7 PLE が発生した関数 (PLE_Window 2048)

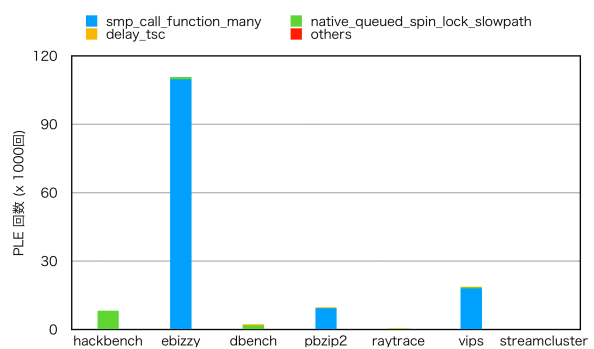


図 8 PLE が発生した関数 (PLE_Window 4096)

わかる。

4.4 PLE の多発要因の考察

PLE は同じ関数で繰り返し何度も発生しており、PLE の発生は連続性があることがわかった。図 11 は PLE が発生した時のトレースポイントによるログである。

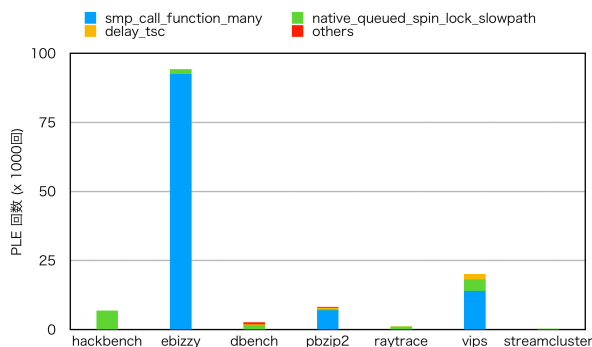


図 9 PLE が発生した関数 (PLE_Window 8192)

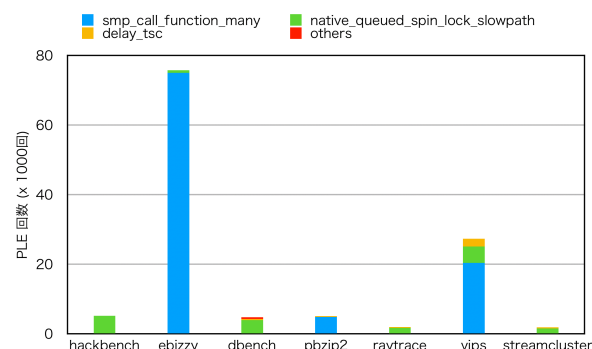


図 10 PLE が発生した関数 (PLE_Window 16384)

PAUSE_INSTRUCTION が原因で、つまり PLE によって VM exit が何度も同一仮想 CPU で連続的に起きていることがわかる。さらに、VM exit 時の rip の値、すなわち IP の値はどれも同じで、同一の関数で PLE が起きていることがわかる。このことから、PLE 発生時、VMM は PLE を起こした仮想 CPU を何度も繰り返しスケジュールしており、ビジーウェイトの要因を取り除くよう適切に仮想 CPU スケジューリングが行われていないと言える。

PLE を避ける仕組みが上手く機能していない原因として、PLE 発生時に KVM によって優先度をブーストされた仮想 CPU と、次に実際にスケジュールされた仮想 CPU が一致しない点が多い点が挙げられる。表 1 はトレースポイントのログから PLE 発生時に KVM によって優先度をブーストされた仮想 CPU と、実際に次にスケジュールされた仮想 CPU の関係を示したものである。なお、プリエンプトされている仮想 CPU がいない場合など、優先度ブーストが行われないこともあるため、合計値が PLE の発生回数よりも小さくなることに留意されたい。この表の通り、優先度がブーストされた仮想 CPU が実際にその直後にスケジュールされるケースは全体の 3.4%だった。つまり、既存の KVM の PLE の多発を避けるための仕組みが上手く機能していないことがわかる。

```
CPU-2663 [000] 549.403790: kvm_entry vcpu: 0
CPU-2663 [000] 549.403803: kvm_my_exit reason
PAUSE_INSTRUCTION rip ffffffff810c7879
CPU-2663 [000] 549.403824: kvm_entry vcpu: 0
CPU-2663 [000] 549.403837: kvm_my_exit reason
PAUSE_INSTRUCTION rip ffffffff810c7879
CPU-2663 [000] 549.403859: kvm_entry vcpu: 0
CPU-2663 [000] 549.403872: kvm_my_exit reason
PAUSE_INSTRUCTION rip ffffffff810c7879
CPU-2663 [000] 549.403892: kvm_entry vcpu: 0
CPU-2663 [000] 549.403906: kvm_my_exit reason
PAUSE_INSTRUCTION rip ffffffff810c7879
CPU-2663 [000] 549.403927: kvm_entry vcpu: 0
CPU-2663 [000] 549.403940: kvm_my_exit reason
PAUSE_INSTRUCTION rip ffffffff810c7879
CPU-2663 [000] 549.403962: kvm_entry vcpu: 0
CPU-2663 [000] 549.403975: kvm_my_exit reason
PAUSE_INSTRUCTION rip ffffffff810c7879
CPU-2663 [000] 549.403996: kvm_entry vcpu: 0
```

図 11 PLE 発生時のログ

表 1 PLE 発生時に優先度ブーストされた仮想 CPU と実際にスケジュールされた仮想 CPU の関係

	同じ (回)	異なる (回)	同じの割合 (%)
dbench	11	1151	0.9
ebizzy	356	6560	5.1
hackbench	2	942	0.2
pbzip2	11	1151	0.0
raytrace	356	6560	0.8
streamcluster	2	942	28.6
vips	2	942	2.3
合計	13174	460	3.4

5. 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[11] 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 [12]。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] [4] [13] . 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 [14] 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 [15] , I/O 処理を行うタスクの実行が不連続になることで 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 [16] などが報告されている。

6. 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 Linux 仮想マシンにおいて、PLE が起こる要因を定量的に調査し、その主な要因を示した。これにより、仮想 CPU スケジューラが PLE の多発を避けるために必要な仮想 CPU スケジューリングの要素を整理した。

また、現在の KVM に備わっている PLE の多発を防ぐための仕組みは上手く機能していないことを示した。

7. 謝辞

本研究は、JST, CREST, JPMJCR1683 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] 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).
- [5] Raghavendra, K.: Virtual Cpu Scheduling Techniques for Kernel Based Virtual Machine (Kvm), pp. 1–6 (online), DOI: 10.1109/CCEM.2013.6684443 (2013).
- [6] : Ebizzy, <https://sourceforge.net/projects/ebizzy/>.
- [7] Bienia, C.: Benchmarking Modern Multiprocessors, PhD Thesis, Princeton University (2011).
- [8] : Hackbench, <http://people.redhat.com/mingo/cfs-scheduler/tools/hackbench.c/>.
- [9] : Dbench, <http://manpages.ubuntu.com/manpages/raring/man1/dbench.1.html>.
- [10] : Pbz2, <https://openbenchmarking.org/test/pts/compress-pbzip2>.
- [11] 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).
- [12] 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).
- [13] 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).
- [14] 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).
- [15] 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).
- [16] 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).