KVM 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 1 つの物理 CPU を複数の仮想 CPU が共有することが一般的である．しかし，ゲスト環境は CPU が常時可動するという想定で実装されている．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，仮想マシンモニタへと制御が移る．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，現状の KVM ではこの機構が十分に機能しておらず， PLE が多発し，性能が大幅に低下することがある．
本研究では PLE が多発する原因として，PLE の多発は原因として， 1) KVM スケジューラと それが依存している Linux スケジューラが上手く連携できていない点, 2)スピンロック中以外の PLE の発生を十分想定せずにヒントを与えている，という 2 つを挙げる．また，それを緩和するための手法を提案し，評価する．その結果，9 つのベンチマークで 0 – 91 % の性能向上を確認できた．

キーワード：仮想化，Pause Loop Exit，スケジューラ，オペレーティングシステム

1. はじめに

仮想環境では, 複数の仮想 CPU が一つの物理 CPU を共有することで, 実行が時間的に不連続となることがある．これを仮想時間の不連続性という．[1]しかし，ゲストオペレーションシステム (OS) は CPU が常時稼働するという前提で実装されており，このギャップが様々な問題を引き起こす.

問題の一つとして，過度に長いビジーウェイトの発生がある．ゲスト OS は短時間で終わるタスクの終了を待つ際はビジーウェイトを用いられる．しかし, 仮想環境では仮想時間の不連続性により，本来短時間で終わるタスクに時間がかかることがある．過度に長いビジーウェイトは CPU 時間を浪費してパフォーマンスを低下させるため，望ましくない.

過度に長いビジーウェイトの影響を緩和するために，Pause Loop Exiting (PLE) というハードウェア支援機構が Intel x86 では提供されている．[3] PLE は過度に長くビジーウェイトしている仮想 CPU を検知し，仮想マシンモニタに仮想 CPU スケジューリングをやり直す機会を与える．この時, 仮想マシンモニタは PLE を引き起こした原因を素早く取り除くよう仮想 CPU をスケジューリングする必要がある. 不適切なスケジューリングは PLE の多発に繋がり，過度に長いビジーウェイトの悪影響を十分に緩和できない.

本研究では，KVM を対象に， PLE 発生時の仮想 CPU スケジューリングが適切かを分析した．その結果, KVM では PLE の多発に繋がる不適切な仮想 CPU スケジューリングを行うことがあるとわかった．また，この分析に基づき, KVM は少ない改変 (41 LOC) で現状よりも PLE の多発を抑え，ビジーウェイトの悪影響を緩和できることを示す.

2. KVM における PLE の活用

KVM では PLE を用いて過度に長いビジーウェイトの悪影響を緩和している．PLE は過度に長いビジーウェイトをする仮想 CPU を検知し，仮想マシンモニタに通知を行うハードウェア機構で，Intel x86 において提供されている.

PLE が起きた時，仮想マシンモニタは PLE の発生要因を素早く取り除くよう仮想 CPU をスケジューリングする必要がある．PLE の発生要因が長時間放置されると，更なる PLE が, 要因が取り除かれるまで延々と発生し, PLE の多発に繋がる．PLE の多発はビジーウェイトにより多くの CPU 時間が消費されることも意味し，望ましくない.

KVM は PLE の多発を避けるよう仮想 CPU スケジューリングを行なっている．[2] 具体的には， PLE の要因をスピンロック操作によるものだと想定し, KVM がスピンロックを保持しうる仮想 CPU を優先的に実行するよう，実際のタスクスケジューリングを行う Linux スケジューラにヒントを与えている.

3. KVM の仮想 CPU スケジューラの分析

KVM における， PLE 発生時の仮想 CPU をスケジューリング手法の有効性について分析した．その結果, KVM はスケジューラの連携不足，そして IPI 処理を要因とする PLE を十分に考慮していないという 2 つの原因で，適切に仮想 CPU をスケジューリングできていないことがわかった.

3.1 スケジューラの連携不足

KVM 上の仮想 CPU はスレッドとして扱われ, 最終的に

1 慶應義塾大学

Linux スケジューラによって実行される．そのため， KVM は PLE を起こした仮想 CPU の代わりに特定の仮想 CPU を優先的に実行したい旨, Linux スケジューラにヒントを与える．しかし，Linux スケジューラは KVM から与えられたヒントよりも，CPU 時間の公平性を重視してタスクをスケジューリングする．そのため，KVM が与えたヒントが十分に活用されないことがある．

3.2 IPI 処理を要因とする PLE の無視

KVM はスピンロック操作を要因とした PLE のみを考慮し，仮想 CPU スケジューリングを行なっている．そのため，PLE はカーネルモードの仮想 CPU のみを対象としていることから，カーネルモードの仮想 CPU のみを優先実行対象としている．しかし，PLE は IPI 操作におけるバリア同期を要因としても起こる．IPI の送信対象はノーマルモードも考えられるため，いつまでも PLE 要因を取り除くための適切な仮想 CPU が実行されないことがある．

4. 提案

3 章で示した問題に対して，それぞれ緩和手法を提案する．

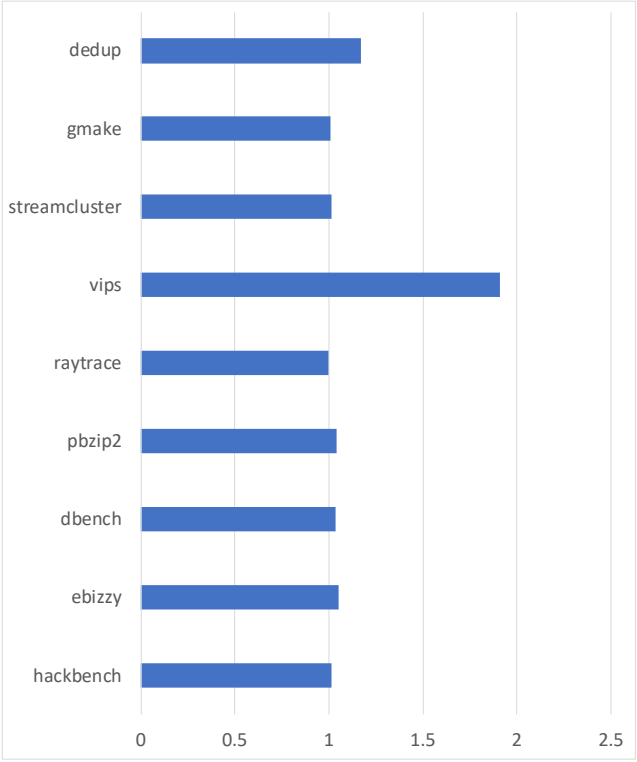
まず，PLE を起こした仮想 CPU が， KVM にヒントとして与えられた優先実行すべき仮想 CPU の実行を妨げている時，PLE を起こした仮想 CPU の CPU 時間による優先度を下げて妨げられないようにする．この時，優先度を下がった仮想 CPU は PLE を起こした仮想 CPU と同一仮想マシン内にあるため，他の仮想マシンに影響を与えることはなく，CPU 時間の公平性を崩すことはない．

次に，PLE 発生時の優先実行すべき仮想 CPU 候補に，カーネルモードの仮想 CPU 以外にも処理待ちの IPI がある仮想 CPU を加える．

5. 評価

本研究では Dell PowerEdge T440 と 10 core 2.2GHz Intel Xeon Silver 4210 の CPU と 8GB のメモリを使用した．なお，ハイパースレッディングはオフとした．ホスト OS には Ubuntu 18.04 LTS with Linux kernel 4.15.18，ゲスト OS には Ubuntu 14.04 LTS with Linux kernel 4.4.134 を使用した．2 つ仮想マシンを作成し，それぞれ仮想 CPU を 10 個，メモリを 2GB 割り当てた．実験は 1 つ目の仮想マシンで CPU 使用率の高い PARSEC swaptions を実行しながら，2 つ目のマシンで各種ベンチマークのパフォーマンスを測定した．使用したベンチマークは ebizzy [4], hackbench[5], dbench[6], pbzip2[7], そして PARSEC [8] と MOSBENCH [9] からいくつか使用した．

グラフ 1 が実行結果を全て対策なしを元に標準化したものである．性能向上は raytrace の時最小で 0% ， vips の時最大で 91%だった．



グラフ 1　提案手法による性能向上

6. 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] Shan, X. Ding and N. Gehani, "APPLES: Efficiently Handling Spin-lock Synchronization on Virtualized Platforms," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 1811-1824, 1 July 2017.
[3] Dong, Y., Mallick, A., Nakajima, J. and Tian, K.: Ex- tending Xen * with Intel Virtualization Technology (2006).
[4] Ebizzy, https://sourceforge.net/projects/ ebizzy/.
[5] Hackbench, http://people.redhat.com/mingo/ cfs-scheduler/tools/hackbench.c/.
[6] Dbench, http://manpages.ubuntu.com/manpages/ raring/man1/dbench.1.html.
[7] Pbzip2, https://openbenchmarking.org/test/pts/ compress- pbzip2.
[8] Bienia, C.: Benchmarking Modern Multiprocessors, PhD Thesis, Princeton University (2011).
[9] Silas Boyd-Wickizer, Austin T. Clements, Yandong Mao, Aleksey Pesterev, M. Frans Kaashoek, Robert Morris, and Nickolai Zeldovich. 2010. An Analysis of Linux Scalability to Many Cores. In Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI '10).