Vol.2017-OS-141 No.22
2017/7/27

CPU処理とI/O処理の高速性を両立するI/O制御機構

田邨 優人[1,a)]　中島 耕太[1,b)]　山本 昌生[1,c)]　前田 宗則[1,d)]

概要：ハードウェアの進歩に伴い，従来よりも大幅に低レイテンシな記憶装置やインターコネクタが登場している．計算機環境のさらなる高性能化，大規模化が求められる昨今では，今後これらのデバイスが主流になると考えられるが，大部分の計算機システムではその高速性を活かせない場合が多い．その一つの原因としてカーネル内でデバイスからの応答の検知にハードウェア割り込みを使用していることが挙げられる．性能が重要視される HPC 分野などではハードウェア割り込みよりも高速にデバイスからの応答検知を行うために polling という手法が用いられる．しかし polling は CPU リソースを占有してしまうという特性から汎用的な計算機システムには積極的に用いられることはなかった．そこで本研究では CPU リソースを管理しながらカーネル内で polling を行うための polling idle ドライバを提案する．提案手法を NVMe over Fabrics に実装して評価を行ったところ，最大 47.1 %のレイテンシ削減効果，最大 77.3 %の iops 向上を確認し，また iops の上限値が従来から 40.2 %向上したことを確認した．

キーワード：I/O polling, kernel land, idle routine, NVMe over Fabrics

1. はじめに

ハードウェアの進歩に伴い，従来よりも大幅に低レイテンシな記憶装置やインターコネクタが登場している．例えば従来の HDD は数 ms のレイテンシであったが，SSD のレイテンシは約 10us であり，また InfiniBand の通信レイテンシも数 us 程である．計算機環境のさらなる高性能化，大規模化が求められる昨今では，今後これらのデバイスがますます主流になると考えられる．しかし，大部分の計算機システムは従来の低速なデバイスを使用することを前提に設計されており，高速なデバイスを用いてもその高速性を活かせていない場合が多い．そのため，高速なデバイスに合わせた計算機システムが求められており，様々な提案がなされている [1], [2], [3], [4].

加えて，ソフトウェアで定義されるインフラストラクチャである Software defined anything (SDx) の進歩が著しい．例えばパケットフォワーディングをソフトウェアで制御する Software defined Network (SDN)，RAID 計算やデータ圧縮やミラーリング処理を行う Software defined Storage (SDS) などがある．これらの技術は一般的にカーネル内で実装されることが多いため，高速なデバイスを活用するにはカーネル内での処理の高速化が不可欠である．

カーネル内処理で改善が求められる処理の一つとして，ハードウェア割り込みが挙げられる．ハードウェア割り込みはデバイスからの I/O 処理終了の応答を受けて CPU に特定の処理を実行させることができる仕組みである．デバイスから応答があるまでは CPU が他の処理を進めることができるため，レイテンシが大きいデバイスを扱う際には性能向上に不可欠な技術であった．しかし，扱うデバイスのレイテンシが低くなるに伴い，ハードウェア割り込みによって発生する，CPU の中断，それに伴うキャッシュフラッシュ，コンテキスト保存・切り換えといった処理によるレイテンシが問題になりつつある．

ハードウェア割り込みよりも高速にデバイスからの応答を検知できる手法に polling というものがある．polling はデバイスの状態を CPU を占有しながら常に監視し続けることでデバイス処理の終了を検知する．ハードウェア割り込みと違って複雑な処理が必要ないため，高速にデバイスからの応答を検知することができ，今後カーネル内処理の高速化のために適用範囲が広がると考えられる [5][6]．ただ，polling という手法はデバイスからの応答を待っている間 CPU リソースを占有してしまうという特性によって積極的に使用されることはなかった．そこで本研究では適切な CPU リソースの管理を行いながら polling を行う手法について提案を行う．また，以降では，第 2 章において本研

[1] 株式会社富士通研究所
Fujitsu Laboratories Ltd, Kawasaki 211-8588, Japan
[a)] tamura.yuto@ip.fujitsu.com
[b)] nakashima.kouta@jp.fujitsu.com
[c)] masao.yamamoto@jp.fujitsu.com
[d)] maeda.munenori@jp.fujitsu.com

究の課題を抽出し，第 3 章で提案手法の説明と提案手法の NVMe over Fabrics への実装方法を述べる．そして第 4 章で FIO ベンチマークを用いた提案手法の評価を行い，第 5 章で関連研究を述べた上で第 6 章でまとめと今後の展開について述べる．

2. 課題

kernel 内で polling を行うにあたって問題となるのは，polling 処理への CPU リソースの割り当てである．まず，各ドライバで行われる複数のオブジェクトへの polling 処理に対してそれぞれ一つずつ CPU コアを割り当ててしまうとリソースが枯渇してしまい，polling を行えないドライバが発生するという問題がある．更に polling が実行されていることによって，システムが本来必要とするタスク処理のためのリソースが供給されないという問題もある．この問題に対し，本研究では以下に示す 2 つの課題を設定する．1，kernel 内における CPU 占有と 2，polling によるタスク処理のスループットの低下である．

2.1 kernel 内における CPU 占有

polling 処理が様々なデバイスドライバで独立に管理され，それぞれが CPU コアを占有している場合，もし polling に必要な CPU リソースが計算機のコア数を超えてしまうと，CPU リソースが使い切られてしまって一部の polling が実行されないという問題が起きる．昨今ではより高速な応答を求める様々なデバイスが増えている．その中で全てのデバイスからの応答に対して高速な検知を実現するために polling を行うには，polling 対象のオブジェクトを一元管理し，それらに対して平等に polling を行う専用ドライバが必要となってくる．

2.2 polling によるタスク処理のスループットの低下

polling はデバイスの応答に対して高速な検知を実現することができるが，デバイスの応答が発生するまでは何らかの処理を進めるわけではない．なので，polling に過剰に CPU リソースを費やしてしまうと，polling 以外のタスク処理の性能が低下してしまう．そのためシステム内に polling 以外のタスクが生じた場合は polling よりも生じたタスクに CPU リソースを渡して実行する方が性能が向上する．polling 以外のタスクに適切に CPU リソースを渡すためには，polling を行っている最中でもシステム内の他のタスクを監視することが必要となってくる．

3. 提案手法

前節において，1，kernel 内における CPU 占有と 2，polling によるタスク処理のスループットの低下の 2 つが課題であることを説明した．本節ではその課題を解決するべく，polling idel ドライバを提案し，その polling idle ドライバを利用するためのモジュール改変の一例として NVMe over Fabrics への対応を説明する．

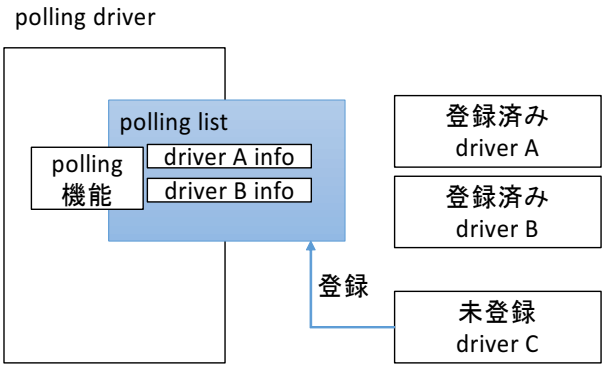


図 1 polling driver 内部の polling list

3.1 polling idle ドライバ

別タスクを監視しながら polling の一元管理を行うために，kernel の idle ルーチンにおいて polling を行う polling idle ドライバを提案する．kernel 内部で polling を行うためには，polling に用いる CPU リソースを管理するために一元管理する必要がある．一元管理を行うために，各ドライバの polling 依頼を受けて，polling を代行する polling ドライバを考える．この polling ドライバはドライバ内部に polling list と呼ぶ，polling をするべきデバイスやオブジェクト，polling 時に実行する関数や検知時に実行するコールバック関数などが記されたリストを持っており (図 1 参照)，このリストは他のドライバに対して公開されている．各ドライバは公開されているこのリストに対して polling に関する情報を書き込むことで polling を依頼する．polling ドライバはこのリストに記載された polling 対象を順々に監視して回る．これによって CPU リソースは各 polling に平等に利用されることになり，常に全てのオブジェクトに対して polling を行うことが可能となる．

また，マルチコアやメニーコア環境において性能を向上させるべく，polling list はその内部に計算機のコア数だけ polling 対象を記載するリストを持つ（図 2 参照）．複数のコアで polling を行う際は，デバイスからの応答を複数のコアで同時に検知してしまうことを防ぐためにロックを用いた排他制御が必要となる．しかし，もし全リストを単一のロックによって排他制御してしまうと，複数コアによる並列性を活かすことができず性能が向上しない．そのため，ロックによる排他制御を行うリストをコアの数だけ作成しておき，polling 対象が登録される際には各リストに均等に登録することで複数コアが並列に polling を行えるようにする．ただ，いくらマルチコアやメニーコアの環境であっても，デバイスからの応答を待機している間は最低限一つのコアが polling を行っていれば十分である．そのため，特定の一つのコアが polling を行っていれば他のコア

図 2　polling list の内部構造

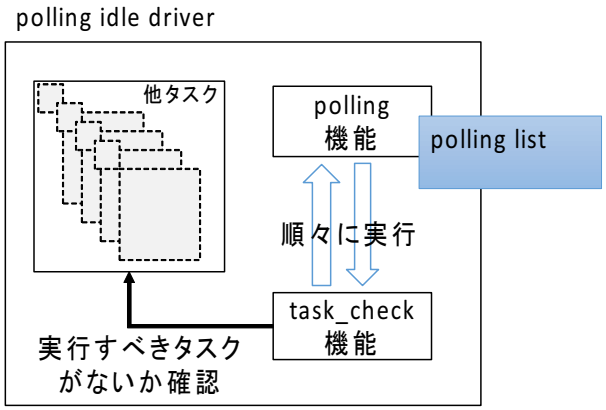


図 3　polling list の内部構造

は polling を行わないように制御する．

また，実行するべき別のタスクに CPU リソースを渡しながら効率よく polling を行うためには，実行可能なタスクを常に把握できることが必要である．そのために idle ルーチンのシステム内のタスクを監視する機能を用いる．通常実行するべきタスクがなくなると CPU は idle ルーチンを繰り返し実行するよう制御される．この idle ルーチンの中では実行するべきタスクが存在するかどうかの確認と，搭載されている CPU ごとの省電力機能の実行が行われる [7]．このうちの実行するべきタスクの確認を行う機能モジュールを利用することで，別のタスクを監視しながら polling を行うことができる．

これらを考慮して提案するのが polling idle ドライバである (図 3 参照)．polling list を基に各デバイスへの polling を一元管理する機能部と idle ルーチンにおけるシステム内のタスク確認機能部を持ち，各 CPU はこの 2 つの機能を順々に実行する．このドライバは cpuidle ドライバの一つとして扱うことができ，起動時にデフォルトの cpuidle ドライバと入れ替えることが可能である．polling idle ドライバが稼働している際には，各デバイスドライバから polling 依頼を行うことが可能となり，CPU が idle 状態になると共にイベント検知の polling 処理が開始される．

3.2　NVMe over Fabrics ドライバの対応

polling idle ドライバに対して polling を依頼するドライバの一例として NVMe over Fabrics を選択する．NVMe over Fabrics は NVMe SSD へのリモート DMA することを可能とするドライバであり，今後 NVMe SSD を搭載した計算機クラスタのノード間通信において主流になると考えられている [8]．

NVMe over Fabrics は通信において verbsAPI を用いて通信資源を生成し，データの送受信を行う．通常の NVMe over Fabrics ではデータの受信イベントを割り込みによって検知し，あらかじめ call back 関数として設定しておいた受信処理によって受信バッファ内のメッセージを処理する．本研究ではここで用いている割り込み機能を使わず，polling idle ドライバが持つ polling リストに対して，受信バッファのアドレスと，受信バッファを確認するための処理，受信バッファ内のメッセージに対して行う処理を登録するよう変更した．これによって，polling idle ドライバは polling リストを基に受信バッファの確認を繰り返し行い，受信バッファ内にメッセージが存在するのを検知すれば受信処理が実行される．

表 1　評価用計算機の仕様

CPU	Intel(R) Xeon(R) E5-2697 v3 @ 2.60GHz
コア数 （スレッド数）	14（14）
L1 キャッシュ	32KB
メモリ	132GB
機種	FUJITSU PRIMERGY RX2540 M1
OS	CentOS Linux release 7.3.1611
カーネル	Linux-4.9

4.　提案手法の性能評価

本研究の課題を提案手法である idle polling ドライバを使用することで解決できるかの評価を行う．

4.1　評価環境

本論文で性能評価に用いた計算機の仕様を表 1 に示す．使用したのは Intel 製 Haswell 世代 Xeon であり，Hyper Threding と Turbo Boost を OFF にして周波数を固定し，性能への外乱の影響を極力抑えた．OS は CentOS を用い，Linux kernel をバージョン 4.9 に入れ替えて使用した．NVMe over Fabrics も kernel 4.9 のバージョンを使用した．この計算機 2 台を Mellanox 製 InfiniBand EDR で接続し，片側を NVMe over Fabrics の initiator ノード，もう片側を target ノードとした．また，target 側のディスクには RAM ディスクを用いた (図 4 参照)．

4.2　評価方法

initiator ノードが target ノード上の RAM ディスクに対して 8KB ランダム write I/O を行う際のレイテンシと iops を FIO ベンチマークにより測定する．測定では同時に I/O を行うスレッド数（numjobs）と 1 スレッドが発行する多重 I/O 数（iodepth）をそれぞれ変化させた．またこの時，ス

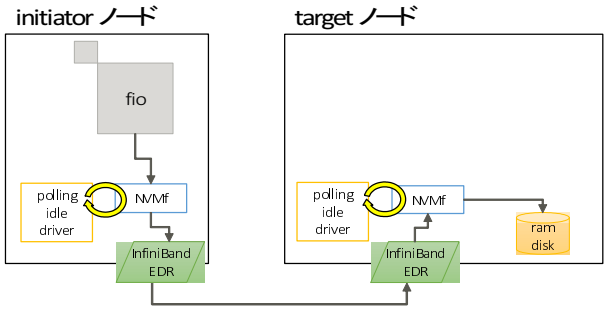
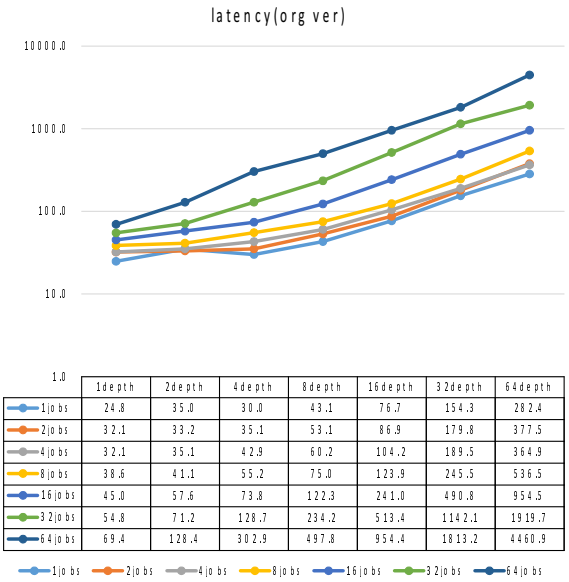
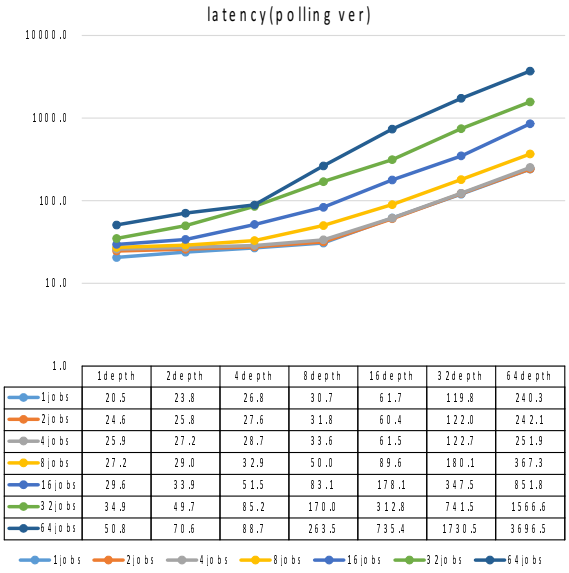


図 4　計算機構成



	1depth	2depth	4depth	8depth	16depth	32depth	64depth
1jobs	24.8	35.0	30.0	43.1	76.7	154.3	282.4
2jobs	32.1	33.2	35.1	53.1	86.9	179.8	377.5
4jobs	32.1	35.1	42.9	60.2	104.2	189.5	364.9
8jobs	38.6	41.1	55.2	75.0	123.9	245.5	536.5
16jobs	45.0	57.6	73.8	122.3	241.0	490.8	954.5
32jobs	54.8	71.2	128.7	234.2	513.4	1142.1	1919.7
64jobs	69.4	128.4	302.9	497.8	954.4	1813.2	4460.9

図 5　オリジナル版レイテンシ



	1depth	2depth	4depth	8depth	16depth	32depth	64depth
1jobs	20.5	23.8	26.8	30.7	61.7	119.8	240.3
2jobs	24.6	25.8	27.6	31.8	60.4	122.0	242.1
4jobs	25.9	27.2	28.7	33.6	61.5	122.7	251.9
8jobs	27.2	29.0	32.9	50.0	89.6	180.1	367.3
16jobs	29.6	33.9	51.5	83.1	178.1	347.5	851.8
32jobs	34.9	49.7	85.2	170.0	312.8	741.5	1566.6
64jobs	50.8	70.6	88.7	263.5	735.4	1730.5	3696.5

図 6　提案手法版レイテンシ

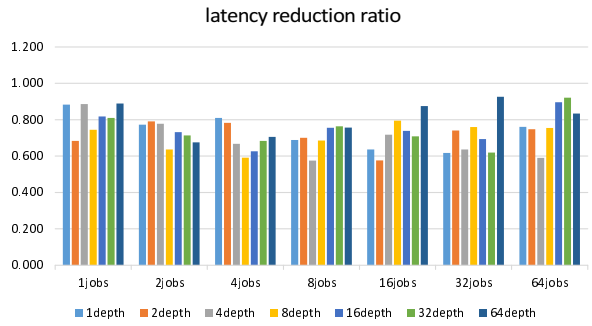


図 7　レイテンシ削減率

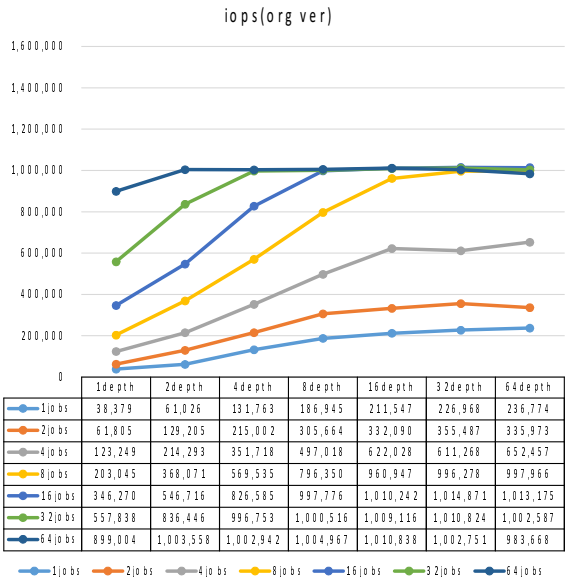
レッド数は計算機のコア数を下回る場合から大きく上回る場合まで設定し，計測を行う．測定はオリジナル版の割り込みによる受信イベントの検知を行う NVMe over Fabrics を使用した場合と，提案手法である polling idle ドライバを適用して polling による受信イベントの検知を行った場合で行い，それぞれの測定結果を比較することで提案手法の評価を行う．

4.3　評価結果

レイテンシの測定結果についてオリジナル版の結果を図 5 に，提案手法適用版の結果を図 6 に，オリジナル版のレイテンシに対する提案手法適用版のレイテンシの比率を図 7 に示す．全ての実験条件においてレイテンシが削減できることを確認し，また最大 47.1 %のレイテンシ削減効果があった．

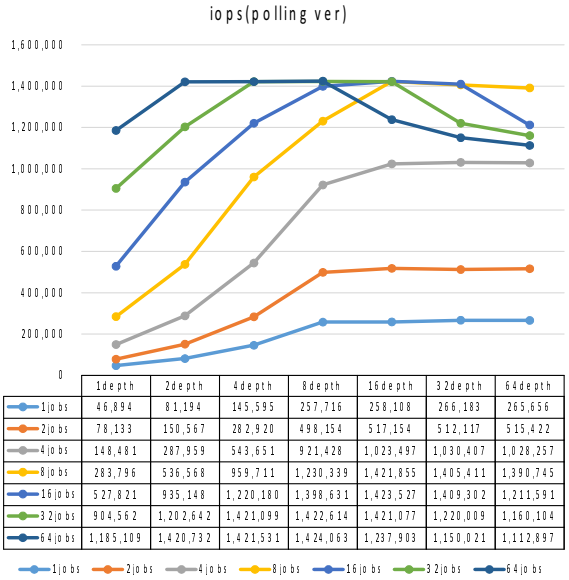
iops の測定結果についてオリジナル版の結果を図 8 に，提案手法適用版の結果を図 9 に，オリジナル版の iops に対する提案手法適用版の iops の比率を図 10 に示す．全ての実験条件において iops が向上することを確認し，最大 77.3 %の iops 向上効果があった．また，iops の上限値がオリジナル版の 1.01Miops から 40.2 %向上して 1.42Miops になったことも確認できる．ただ，スレッド数 (numjobs) とスレッド当たりの多重 I/O 数（iodepth）の積が 512 から 1024 以上に変化するところで性能の低下が起きていることからまだ性能改善が見込めることが分かる．

上記の結果から，I/O を行うスレッド数が計算機のコア数である 14 を超えても性能が向上していることが確認できる．通常の polling によるイベント検知であれば，計算機のコア数を超える数のスレッドが polling を行おうとした際には，polling にコアが占有されすぎてしまう．そのため，一部のスレッドのイベント検知処理やベンチマークプロセスの処理が滞ってしまい，性能の低下を引き起こしてしまう．しかし，提案手法では polling を行うコアを polling idle ドライバで統一的に管理しており，また実行可能なタスク（ベンチマークプロセス）の存在を確認しながら polling を行っているため，polling 用にコアを占有しすぎてしまうことがない．そのため，計測結果のように全ての実験条件において polling による性能向上が確認できる．



	1depth	2depth	4depth	8depth	16depth	32depth	64depth
1jobs	38,379	61,026	131,763	186,945	211,547	226,968	236,774
2jobs	61,805	129,205	215,002	305,664	332,090	355,487	335,973
4jobs	123,249	214,293	351,718	497,018	622,028	611,268	652,457
8jobs	203,045	368,071	569,535	796,350	960,947	996,278	997,966
16jobs	346,270	546,716	826,585	997,776	1,010,242	1,014,871	1,013,175
32jobs	557,838	836,446	996,753	1,000,516	1,009,116	1,010,824	1,002,587
64jobs	899,004	1,003,558	1,002,942	1,004,967	1,010,838	1,002,751	983,668

図 8　オリジナル版 iops



	1depth	2depth	4depth	8depth	16depth	32depth	64depth
1jobs	46,894	81,194	145,595	257,716	258,108	266,183	265,656
2jobs	78,133	150,567	282,920	498,154	517,154	512,117	515,422
4jobs	148,481	287,959	543,651	921,428	1,023,497	1,030,407	1,028,257
8jobs	283,796	536,568	959,711	1,230,339	1,421,855	1,405,411	1,390,745
16jobs	527,821	935,148	1,220,180	1,398,631	1,423,527	1,409,302	1,211,591
32jobs	904,562	1,202,642	1,421,099	1,422,614	1,421,077	1,220,009	1,160,104
64jobs	1,185,109	1,420,732	1,421,531	1,424,063	1,237,903	1,150,021	1,112,897

図 9　提案手法版 iops

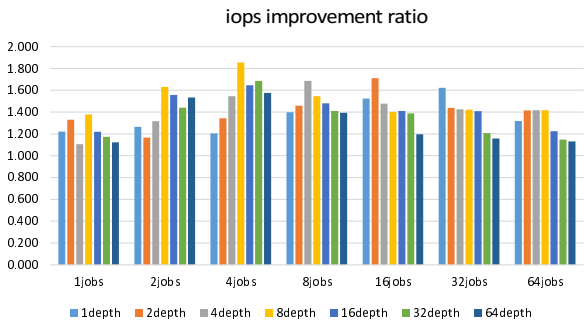


図 10　iops 向上率

5. 関連研究

従来よりも高速なデバイスが登場し，それらの性能を活かすためにデバイスからの応答検知に polling を用いる研究が数多くなされている．Dong ら [9] は不必要な polling やメモリ間のチャネル競合を防ぐべく，デバイスのレイテンシに応じて動的に polling の間隔を変化させる Dynamic Interval Poiing を提案している．本研究でも性能向上に寄与しない不必要な polling は防ぐべきであると考え，それらの CPU リソースを別のドライバに対する polling や他のタスク処理に割り当てるための polling idle ドライバを提案した．

Yu ら [10] は，OS が高速なストレージデバイスを活用するための 6 つの最適化を提案しており，その一つが割り込みの代わりに polling を用いるというものである．本研究でもカーネル内において polling を行う手法を提案しており，更に複数ドライバでの polling を idle ドライバで一元管理することを提案している．

6. おわりに

我々は本研究において他のタスクを監視しながら kernel 内において polling の一元管理を行うために，idle ルーチンにおいて polling を行う polling idle ドライバを提案した．これを linux kernel 上の idle ルーチンと，NVMe over Fabrics に実装し，InfiniBand EDR をインターコネクタに用いて，FIO による 8KBrandom write のベンチマークを行ったところ，I/O を発行するスレッド数が計算機のコア数を上回る条件においても性能が向上することを確認できた．また，提案手法を適用しなかった場合と比較して，最大 47.1 %のレイテンシ削減効果，最大 77.3 %の iops 向上を確認し，また iops の上限値が 1.01Miops から 40.2 %向上して 1.42Miops を達成したことを確認した．これは提案手法によって従来手法よりもデバイスの素性能を引き出すことができていると言える．計算環境の発展はハードウェアの進歩のみでなく，ハードウェアを制御して性能を引き出すソフトウェアの進歩も不可欠である．しかし，汎用的な計算環境においてはリソースの柔軟性についての問題を解決しない限り，新しい技術が取り込まれることはない．本研究は polling 技術について，CPU リソースの柔軟性の一問題を解決するものであり，今後は更なるリソース柔軟性についての問題に取り組んでいく．具体的には，I/O 処理とは異なる種類のベンチマークを同時に実行した際の性能分析，NVMe ドライバやその他ドライバへの対応，複数種類のデバイス，複数ドライバ間で polling を行った際の評価を行うことで更なる課題を見つける．

参考文献

[1] : Nvm express revision 1.3, NVM Express Inc, (online), available from ⟨http://www.nvmexpress.org/wp-content/uploads/NVM_Express_Revision_1.3.pdf⟩ (2017).
[2] Caulfield, A. M., Mollov, T. I., Eisner, L. A., De, A.,

Coburn, J. and Swanson, S.: Providing Safe, User Space Access to Fast, Solid State Disks, *SIGARCH Comput. Archit. News*, Vol. 40, No. 1, pp. 387–400 (online), DOI: 10.1145/2189750.2151017 (2012).

[3] Wei, M., Bjørling, M., Bonnet, P. and Swanson, S.: I/O Speculation for the Microsecond Era, *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, Philadelphia, PA, USENIX Association, pp. 475–481 (online), available from ⟨https://www.usenix.org/conference/atc14/technical-sessions/presentation/wei⟩ (2014).

[4] Kim, H.-J., Lee, Y.-S. and Kim, J.-S.: NVMeDirect: A User-space I/O Framework for Application-specific Optimization on NVMe SSDs, *Proceedings of the 8th USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage'16, Berkeley, CA, USA, USENIX Association, pp. 41–45 (online), available from ⟨http://dl.acm.org/citation.cfm?id=3026852.3026861⟩ (2016).

[5] Yang, J., Minturn, D. B. and Hady, F.: When Poll is Better Than Interrupt, *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST'12, Berkeley, CA, USA, USENIX Association, pp. 3–3 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2208461.2208464⟩ (2012).

[6] Liu, J. and Abali, B.: Virtualization Polling Engine (VPE): Using Dedicated CPU Cores to Accelerate I/O Virtualization, *Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, New York, NY, USA, ACM, pp. 225–234 (online), DOI: 10.1145/1542275.1542309 (2009).

[7] Pallipadi, V.: cpuidle - Do nothing, efficiently..., (online), available from ⟨http://ols.108.redhat.com/2007/Reprints/pallipadi-Reprint.pdf⟩ (2007).

[8] : Nvm express over Fabrics revision 1.0, NVM Express Inc, (online), available from ⟨http://www.nvmexpress.org/wp-content/uploads/NVMe_over_Fabrics_1_0_Gold_20160605-1.pdf⟩ (2016).

[9] Shin, D. I., Yu, Y. J., Kim, H. S., Choi, J. W., Jung, D. Y. and Yeom, H. Y.: Dynamic Interval Polling and Pipelined Post I/O Processing for Low-Latency Storage Class Memory, *Presented as part of the 5th USENIX Workshop on Hot Topics in Storage and File Systems*, San Jose, CA, USENIX, (online), available from ⟨https://www.usenix.org/conference/hotstorage13/workshop-program/presentation/Shin⟩ (2013).

[10] Yu, Y. J., Shin, D. I., Shin, W., Song, N. Y., Choi, J. W., Kim, H. S., Eom, H. and Yeom, H. Y.: Optimizing the Block I/O Subsystem for Fast Storage Devices, *ACM Trans. Comput. Syst.*, Vol. 32, No. 2, pp. 6:1–6:48 (online), DOI: 10.1145/2619092 (2014).