メニーコア向け高速トランザクション処理技術

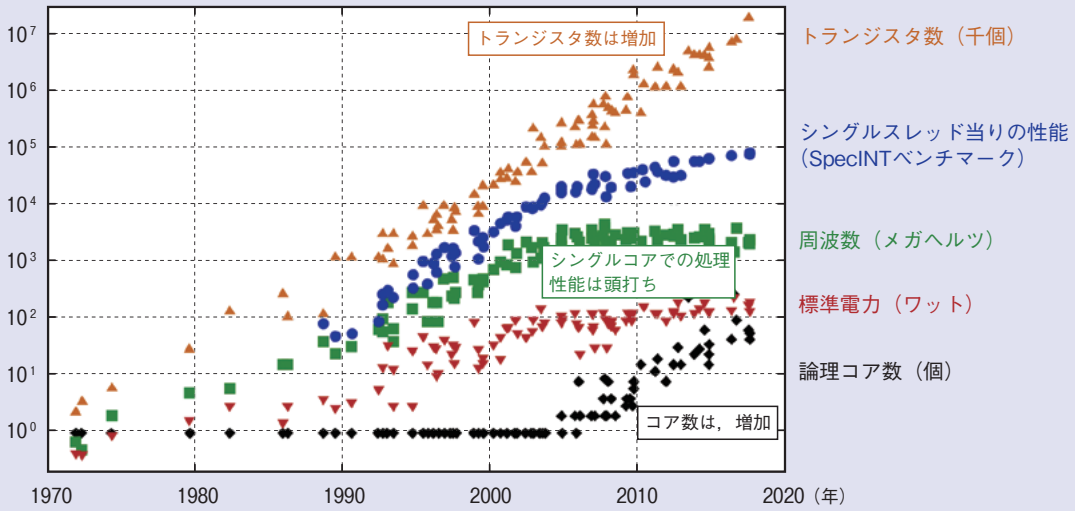
IoT（Internet of Things）やFinTech（Finance & Technology）などの分野において，新しいサービスが提供されています．これらのサービスが提供するAPI（Application Programming Interface)を，マシン間やサービス間において自動的に呼び出すことで，さらに多様なサービスが生まれています．その結果，データベースの分離性を担保したまま，読込・更新・削除を行う処理が年々増加しており，その傾向は今後も続いていくと予測されています．本稿では，これらの処理を高速に行うための，メニーコアCPU上でスケールする高速トランザクション処理技術について紹介します．

なかぞの　しょう　うちやま　ひろゆき
中園　翔　/内山　寛之

NTTソフトウェアイノベーションセンタ

膨大なデータベース処理と技術的課題

IoT（Internet of Things）においてモノどうしがやり取りし合う状況や，複数のWebサービスが連携し，自動でAPI（Application Programming Interface）を呼び出し合うなど，機械やプログラムがその他のサービスを呼び出すM2M（Machine to Machine）が台頭しています．結果として，これまでにないほどの膨大なデータベース処理が発生しています．近年，CPU上のトランジスタ数はムーアの法則にしたがって増加していますが，それらはCPUのコア数を増加させることで実現されてきました(**図1**)．既存のデータベースでは，このようなメニーコアCPUを想定してつくられていないため，コア数を増やすとむしろデータベース処理のスループットが減少する例が報告されています[(1)]．増加するデータベース処理に対応するため，読込処理が大半を占めるユースケースに



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp
This chart is provided under the permissive 'Creative Commons Attribution 4.0 International Public License'
Adjusting points are adding comments and translating the subject and the legends.
2010年までのデータとその図示は，M.Horowitz, F. Labonte, O. Shacham, K.Olukotun, L.Hammond, and C. Battenによって実施された．
2010-2017年のデータとその図示は，K. Ruppによって実施された．この図は，クリエイティブコモンズライセンス4.0に基づき，提供されている．
オリジナルの図に対して，翻訳とコメントを付加した．
出典：https://github.com/karlrupp/microprocessor-trend-dataを基に作成．

図1　マイクロプロセッサのトレンドデータ（42年間）

対しては，メニーコアCPUにおいてもスケールするSiloといった研究プロダクトが提案されています[(2)]．しかし，これらのプロダクトは，更新処理が多いユースケースにおいては，スケールしないことが知られています．IoTを用いたサービスが増加するにつれて，モノの位置や温度といったセンサ情報や数万点にものぼるサプライチェーンマネージメントの状態を逐次更新する必要が生じています．

また，キャッシュレス決済や小額決済，小額の送金等の処理においても，更新処理量の増加が見込まれています．これらの処理は，トランザクションの分離性[*1]を担保しながら更新される必要があります．各CPUコアが並列して処理する際に，同じデータ群に対する読み書きを，1つずつ処理することで分離性を担保して処理を行うことができます．しかし，このような処理の仕方では，各CPUコアは，お互いの処理が終わるのを待ってから処理をするためにスケールせず，データベース処理のスループットは減少してしまいます．

図2では，データベース処理に利用される既存のアルゴリズムが，CPUコア数に対してトータルスループットがどのように変化するかを示しています．コア数が32程度でスループットの上限を迎え，さらにコア数を増加させていくと全体のスループットが落ち込んでいくことが観測されます．そのため，更新処理の多いユースケースに対しては，データベースが利用されている現場においては，データベースの分離レベル[*2]を緩和することで，処理性能を高速化するなどの対応が取られています．しかし，このような対応では，危険性がつきまといます．例えば，あるビットコインの交換所では，分離レベルを下げたデータベースを用いたために，ハッカーから攻撃を受け，交換所にある通貨をすべて引き出されてしまい，交換所が閉鎖に追い込まれるという事態となりました．必ずしも事業が停止するほどのインパクトがあるわけではないにせよ，実際に処理したデータが分離性を持たない場合，何らかの影響を事業やユーザに与える可能性があります．以上のことから，データベースの分離性を担保したうえで，メニーコアCPU上における読込・更新・削除処理を高速化することは，サービスを安定かつ低コストで提供するために極めて重要な課題となります．

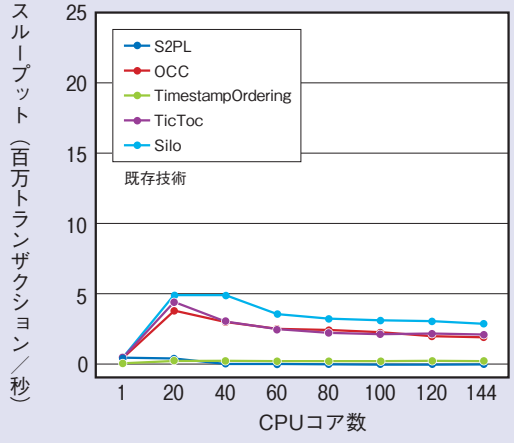


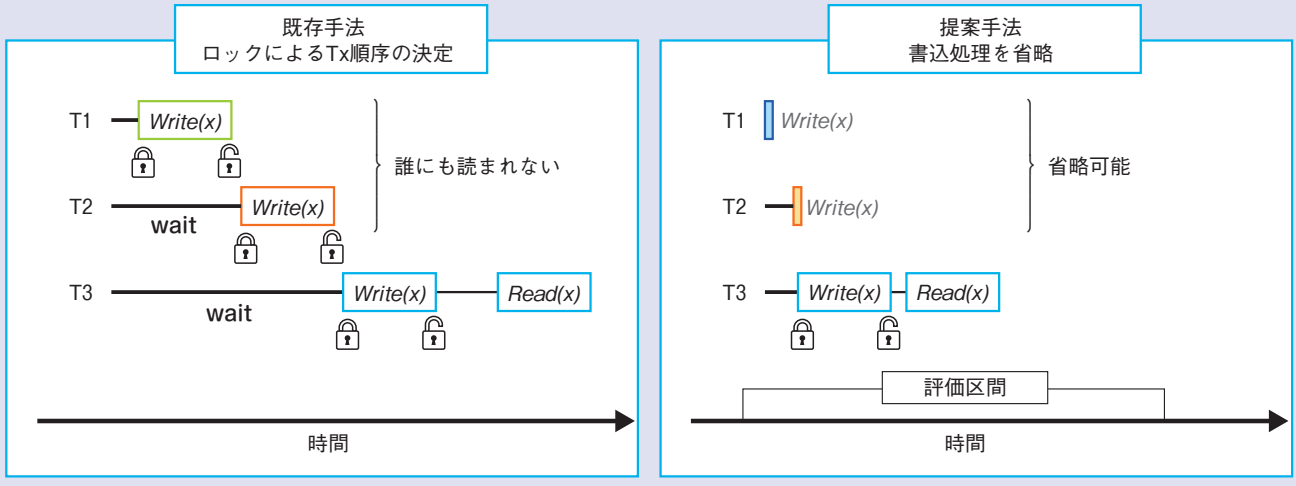
図2　既存技術のメニーコアCPU上での性能

＊1　分離性：並列して実行されるトランザクション間で，トランザクションが行う操作の過程が他のトランザクションから隠蔽されること．分離性には，レベルが存在し，分離レベルと呼びます．シリアライザブルと呼ばれる分離レベルは，もっとも強いものであり，並列実行しているにもかかわらず，あたかも1つずつ順番に処理が行われているのと同じ結果を得ることができます．本提案技術では，シリアライザブルを保証しています．

＊2　分離レベル：比較的緩いレベルであるread committedを採用すると，処理性能は早くなりますが，同一トランザクション内で複数回の同じ読み取りを行うときに結果が違うなどといった問題が発生することが知られています．

提案手法

前述のように，読込の多いユースケースにおいては，いくつかの高速処理手法が提案されてきましたが，今後想定される更新処理の多いユースケースに対しては，処理を高速化することができませんでした．1つのデータに対して，更新処理を複数のコアが並列して実行した場合，どちらかが必ず待つ必要があるためです．この課題に対し，NTTソフトウェアイノベーションセンタでは，Invisible Writeと呼ばれる新たな手法を開発し，更新処理の大幅な高速化を実現しました．本手法では，「誰にも読まれないならば，更新処理を行う必要がない」ことに着目し，もっとも安全な分離レベルを担保しつつ，更新処理の入替を行い，誰にも読まれない更新処理をつくり出すことに成功しました．既存手法と提案手法の違いを**図3**に示します．図中のT1，T2，T3は，データベース処理を並列に処理するプロセスを示しており，xは更新処理の対象を示しています．Write(x)はxに何らかの値を書き込むことを示します．横軸は時間の経過を表しています．既存手法では，更新処理を行う際には，書込対象に対してロックを取得します．そのため，T1の処理が終わらない限り，T2の処理は始まりません．また，T2の処理が終わらない限り，T3の処理は終わらないことを示しています．このとき，T1，T2によって更新されたデータは，誰からも読み込まれないため，実質的に書込を省略することができます．図2の提案技術では，ある一定期間の中で読み込まれない更新処理を特定し，その処理を省略することで高速化していることを示しています．さらには，多版型同時実行制御の理論を応用し，トランザクション群の書込を行う順番を入れ替えることで，省略可能な書込を生み出しています．以上のことから，提案手法では，これまでのデータベースでは省略できなかった更新処理を省略可能とすることができ，膨大な更新処理が発生した場合においても，高速に処理することが可能となっています．**図4**では，既存技術に私たちの提



(a) ロック期間が長いほどスループットが低下　　(b) 省略可能なほどスループットが向上

図3　既存手法と提案手法の違い

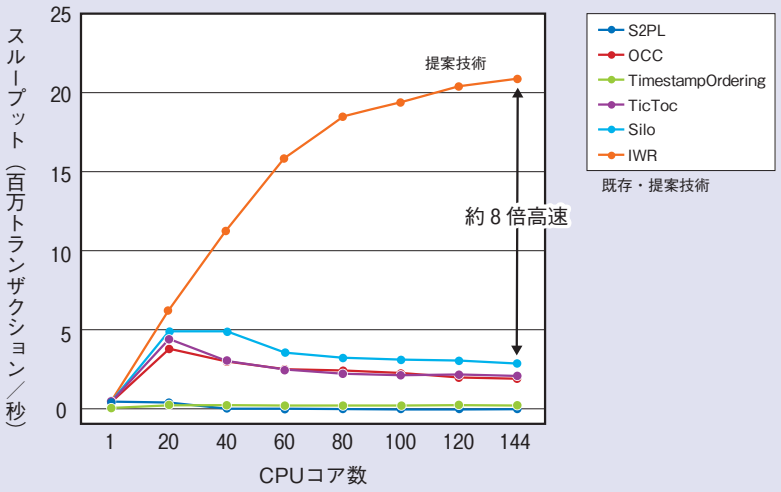


図 4　提案手法と既存手法の比較

案手法をプロットしたものとなっており，提案技術ではCPUコア数が増加するにつれて，全体の処理スループットも増加していることが分かります．144コアにおいては，既存のアルゴリズムに比べて 8 倍高速化しており，秒間2000万トランザクションもの処理を実現しています．この処理速度は，1.7兆トランザクション/dayであり，今後想定されるAPIの呼び出し量 1 兆トランザクション/day[(3)]を十分に処理できる処理能力となっています．

＊3　SQL：標準化されているデータベースに対する問い合わせ言語．
＊4　O/R Mapper：オブジェクト指向言語において，オブジェクトの値を更新した際に，自動的にデータベースへ反映され，読込時には自動的にデータベースから読み込むという機能を提供するソフトウェアライブラリの総称．

今後の展開

本技術は，組み込みデータベースとして利用いただくべく開発を進めています．インタフェースとしては，キーバリューストアに近いものを用意する予定であり，読込と書込を複数のキーに対して，一度に処理することが可能となります．本技術をさまざまな分野に組み込んでいただくことで，最新のハードウェア上でも十分に性能を引き出すことのできるアプリケーションを開発することが可能となります．さらに，キーバリューストアとしてのインタフェース以外にも，SQL[*3]（Structured Query Language）やO/R（Object-Relational）Mapper[*4]等を用意することで，より多くのユーザにご利用いただけるように開発を進めていきたいと考えています．

■参考文献

(1) X. Yu, G. Bezerra, A. Pavlo, S. Devadas, and M. Stonebraker: “Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores,” Proc. of VLDB Endowment, Vol.8, No.3, pp.209-220, 2014.
(2) S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden: “Speedy transactions in multicore in-memory databases,” SOSP 2013, Farmington, U. S. A., pp. 18-32, Nov. 2013.
(3) https://www.ae.be/blog-en/api-billionaires-club-about-to-welcome-trillionaires-members

（左から）中園　翔/ 内山 寛之

今後増えると予測されるさまざまな新しいサービスを支えるため，安全かつ高速なデータベース処理技術の研究開発を進めていきます．事業におけるニーズや要件を議論しながら，良いものをつくっていきたいと考えています．

◆問い合わせ先
NTTソフトウェアイノベーションセンタ
分散処理基盤技術プロジェクト
TEL　0422-59-6201
FAX　0422-59-3739
E-mail　hiroyuki.uchiyama.fy@hco.ntt.co.jp