社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

並列データベースシステムに於ける RDMA を用いたリモート入出力性能の検討

加藤　滉貴†　小沢　健史††　合田　和生††　喜連川　優††,†††

† 東京大学大学院情報理工学系研究科　〒113–8656 東京都文京区本郷 7-3-1
†† 東京大学生産技術研究所　〒153–8505 東京都目黒区駒場 4-6-1
††† 国立情報学研究所　〒101–8430 東京都千代田区一ツ橋 2-1-2
E-mail: †{kokik,ozawa,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

あらまし　並列 DBMS のストレージアーキテクチャとしては，伝統的に Shared nothing や Shared storage が用いられてきた．しかし，近年のコンバージドネットワークなどの高速ネットワーク技術を背景に，「Disaggregated Storage」と呼ばれる新しいストレージアーキテクチャが登場した．これは，Shared nothing と Shared storage の両方の利点を生かしたハイブリッドなアーキテクチャであるが．サーバに対するデータの局所性が存在するので，Disaggregated Storage 上に高性能な DBMS を構築するためには，このアフィニティ問題を解決する必要がある．本研究では，Disaggregated Storage 上のローカル/リモート IO の潜在的な能力を調査し，オペレータと IO のアフィニティ制御のアルゴリズムを考案した．その結果，ローカル IO の約半分の IOPS に相当する 230 万 IOPS のリモート IO に成功した．
キーワード　並列 DBMS，入出力，ネットワーク，RDMA，InfiniBand

A Study on Remote I/O Performance Using RDMA in a Parallel Database System

Koki KATO†, Ozawa TSUYOSHI††, Kazuo GODA††, and Kitsuregawa YU††,†††

† Graduate School of Information Science and Technology, The University of Tokyo
7–3–1 Hongo, Bunkyo-ku, Tokyo 113–8656, Japan
†† Institute of Industrial Science, The University of Tokyo
4–6–1 Komaba, Meguro-ku, Tokyo 153–8505, Japan
††† National Institute of Informatics
2–1–2 Hitotsubashi, Chiyoda-ku, Tokyo 101–8430, Japan
E-mail: †{kokik,ozawa,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

1. はじめに

DBMS は様々な観点から性能向上につながる進化を遂げてきた．例えば，近年の DRAM の高容量化・低コスト化に伴うインメモリ DBMS の登場である．近年の分散インメモリ DBMS に大きな貢献をもたらした Silo [1] は，既存の伝統的な商用 RDBMS である PostgreSQL より 200 倍以上速いという指摘も出ている [2]．このように DBMS の性能が向上する一方で，そのネットワーク部については，あまり改善が行われてこなかった．インターネットにおいてもデファクトスタンダードの通信技術である TCP/IP が，DBMS のようなデータセンタ内のノード間通信においてもデファクトスタンダードで，これは今尚数十年続いている傾向である．TCP/IP は幅広いアーキテクチャ上で信頼性の高い通信路を提供する一方で，パフォーマンスの観点からは DBMS に対して十分に最適化されているとはいえず，しばしばネットワークが DBMS においてボトルネックとなることが多い．実際に，ネットワークがボトルネックであるという認識のもと設計されている DBMS も多い [3].

しかし近年，コンバージドネットワークの登場によって，ネットワークの性能が向上し，ストレージの通信路とネットワークの通信路のパフォーマンスの差が少なくなった．例えば，コンバージドネットワークとして構成された InfiniBand を採用し

た DBMS において，Remote Direct Memory Access (RDMA) と呼ばれる高スループット・低遅延なノード間通信を可能とする技術を導入しようという研究動向が存在する [2], [4]〜[19]. このように DBMS の構成要素のパワーバランスは年々従来とは異なった様相を呈してきている．

大規模データを高速に DBMS で処理するための手段として，並列 DBMS を構成して，負荷を複数のマシンで分散するというものが古くからよく使われる．この並列 DBMS のストレージアーキテクチャとして，Shared-nothing と Shared-storage という 2 つがあった．（第 2.1 節参照）しかし，前述のようにコンバージドネットワークの登場によって，ネットワークスタックが高速化した結果，Disaggreated storage という新しいストレージアーキテクチャが考案されている．

Disaggreated storage とは，Shared-nothing と Shared-storage の利点を併せ持つハイブリッドなアーキテクチャであるが，サーバに対してストレージの局所性が存在するため，潜在的には非常に高いパフォーマンスを持つ一方で，局所性を上手に利用しないとかえってパフォーマンスが落ちてしまうというアフィニティの問題が存在し，この問題は現在未解決である．

そこで本研究では，ローカル IO とリモート IO の実験的考察を行い局所性を明らかにする．

本論文の構成は以下のとおりである．第 2. 節で並列 DBMS のストレージアーキテクチャ・RDMA を用いたリモート入出力について述べる．第 3. 節で，測定方法・実験環境・実験結果について述べる．第 4. 節で，関連研究を紹介する．第 5. 節で最後に本論文を総括する．

2. 並列 DBMS のストレージアーキテクチャと RDMA を用いたリモート入出力

2.1 並列 DBMS のストレージアーキテクチャ

並列 DBMS をストレージアーキテクチャの視点から見ると，ストレージをマシン間で共有するか否かという観点で，図 1 のように Shared-nothing, Shared-storage の 2 つのアーキテクチャがそれぞれ 1980 年代，2000 年代から存在する．

Shared-storage アーキテクチャでは，複数のマシンで一つのグローバルなストレージを共有しており，論理的にはシンプルである．

一方で，Shared-nothing アーキテクチャでは，各マシンはリモートマシンのストレージにはアクセスできない．それ故にデータを格納する際は，データを分割して各ストレージに保管する方法（パーティショニング）や，何らかの方法によりストレージ間で同期を取り全てのストレージにデータのコピーをもたせる方法などを取る．

これらの伝統的なストレージアーキテクチャに対して，第 1. 節で述べたように Disaggretaed storage が存在する（図 2）.

2.2 Remote Direct Memory Access (RDMA)

Remote Direct Memory Access，通称 RDMA とはその名の通り，リモートマシンに存在するメモリ上に読み書きを行う技術である [7], [13], [19]. つまり，TCP/IP が提供するソケットインタフェースのようには抽象化されておらず，RDMA は通信

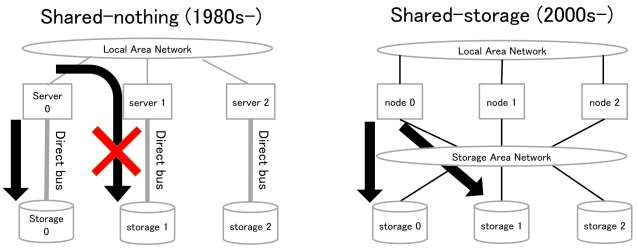

図 1: 既存のストレージアーキテクチャ

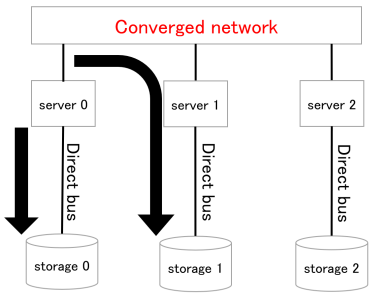

図 2: Disaggregated storage

先のメモリに書き込むところまでしか機能として提供していない．そこから先の，どのようにどのタイミングで通信先のアプリケーションがその書き換えられた内容を見るかといった処理は RDMA のユーザ側に任されている．

本研究ではコンバージドネットワークを構成する InfiniBand 上の通信プロトコルとして採用する．

2.3 RDMA を用いたリモート入出力

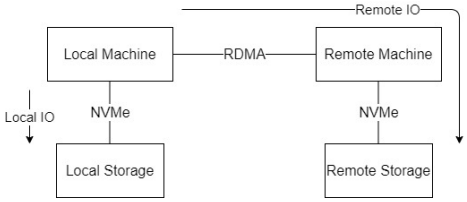

図 3: RDMA を使用した並列 DBMS のアーキテクチャ

本研究におけるリモート入出力の概要を説明する．図 3 のように，2 台のマシンがあったときに，片方からもう一方のストレージにアクセスを行うことが本研究におけるリモート入出力の想定である．RDMA を介してリモート入出力を行う方法であるが，リモートマシンに RDMA を介して入出力命令を発行し，リモートマシン上のプロセスがその命令を受けてさらに自マシンのストレージ（つまり Remote Storage) に入出力命令を発行する．そして結果を受け取ったリモートマシン上のプロセスは，往路と同じように RDMA を介して結果をローカルマシンに送り返すことで入出力が実現される．

リモート入出力における具体的なメッセージ内容とそのサイズは図 4 のとおりである．ローカルマシンにおいて，読み出したい Remote Storage のアドレスが決定したら，そのアドレス値を RDMA を用いてリモートマシンに送信する．アドレス値を受けとったリモートマシンのプログラムは，pread システ

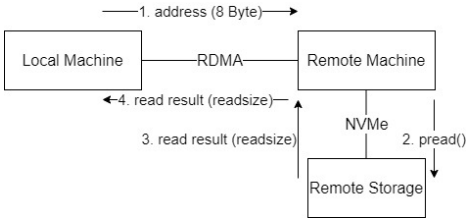

図 4: リモート入出力のフロー

ムコールを用いて Remote Storage 上のデータを取得する．取得したデータはメモリ上に展開されるため，この領域をそのまま RDMA でローカルマシンに送信することで，ローカルマシンは Remote Storage のデータを取得できリモート入出力が完了する．

2.4 リモートマシンの入出力スレッド管理

リモートマシンは，RDMA により読むべきアドレス値を逐次受信している．RDMA の受信確認はポーリングにより行うが，そのポーリングを行う while ループの中で，入出力を発行するとブロックされてしまう．それを防ぐため，入出力スレッドは RDMA 受信確認ポーリングスレッドとは別に用意する．その方式は，動的にスレッドを生成するものと，静的にスレッドを用意しておくものの 2 つである．

動的に入出力スレッドを生成する方式においては，RDMA によりアドレス値を受信したら，pthread_create() により入出力スレッドを新たに生成する．その入出力スレッドの中で read() 等の入出力命令を発行する．

静的に入出力スレッドを用意しておく方式では，ワークロードの開始前に予め入出力スレッドを用意しておき，RDMA によりアドレス値を受信するたびに，シグナルを用いてスレッド間通信を行うことで，RDMA ポーリングスレッドは入出力スレッドに入出力命令を委譲する．

3. 評 価 実 験

3.1 測 定 方 法

ワークロードとしては 512Byte ランダムリードを行う．Remote Storage に対してランダムリードを大量に発行し，その実行時間から IOPS を測定する．即ち読みたい Remote Storage のアドレス値を乱数生成器によりローカルマシンで生成し，そのアドレス値に対するリモート入出力を大量に行う．またリモート入出力はローカル入出力の何割の性能を出せるのかという観点は実用上においても非常に重要であるので，ベースラインとして Local Storage への入出力性能を測る実験を行い，Remote Storage に対する入出力性能と比較する．

また，別のベースラインとして純粋な RDMA のネットワーク性能を測る実験を行った．具体的には，リモート入出力では本来，アドレス値をリモートマシンに送った後，リモートマシンはストレージに read 命令を発行するが，今回の実験では read 命令を発行せずに，既にメモリ上に存在しているダミーデータを read の結果としてローカルマシンに送り返すワークロードを行った．つまり図 4 において 2,3 の工程は省略し 1,4 のみの工程を行うものとなっている．リモートマシンは read を行わずに，ただダミーデータを送り返すだけなので本論文ではこのワークロードを RDMA 通信と呼ぶ．この場合入出力は発行していないので，IOPS に対応する指標として OPS を用いる．

リモート入出力は，ローカル入出力と RDMA 通信を接続したものであるので，その性能は 2 つのうち低い方に律される．

またランダムリードの際は，一つのリモート入出力を発行してその結果が返ってくるのを待ってから，次のリモート入出力を行うのではなく，結果が返ってくるのを待たずに次々にリモート入出力を発行し，非同期に結果が返ってくるようにしており，多重的に入出力を発行することで，リモート入出力のレイテンシを隠蔽し IOPS の向上を測っている．

また実験に使用したプログラムでは，性能向上のためコネクションを複数張れるようにした．コネクションとはマシン間で RDMA 通信を行うときに張るもので，一つのコネクションに送信と受信をそれぞれ受け持つ 2 つの専属のスレッドがローカルマシンとリモートマシンでそれぞれ付くようにプログラムした

3.2 実 験 環 境

表 1: 実験環境

サーバ	Supermicro Super Server
CPU	Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz
メモリ	128GiB
SSD	Intel SSDPE2MD800G4 ×10
HCA	Mellanox ConnectX-3
InfiniBand	56Gbps
OS	CentOS Linux release 7.6.1810 (Core)

本研究の実験環境は 2 台のマシンから構成される．これらのマシンは RDMA を提供するネットワークチャネルのデファクトスタンダードとなっている InfiniBand によって接続される．両マシンの仕様は同一である．詳しい環境を表 1 に示す．

またストレージには NVMe 接続された SSD を用いる．ストレージへのアクセス速度が遅いとシステムのボトルネックがストレージになってしまい，正しく RDMA の性能を測ることができなくなってしまう．そこで本研究では，十分に性能が見込めるストレージ構成をとっている [20].

3.3 実験の方法と結果

3.3.1 ローカル入出力の性能測定

リモート入出力に対するベースライン実験として Local Storage に対してランダムリードを行う実験を行った．つまりこの実験ではマシンは 1 台しか使用せず，RDMA 等のネットワークプロトコルは一切使用していない．実験設定としては，スレッド数を独立変数として設定して実験した．n スレッドを立ち上げ，その各スレッドで乱数生成→その乱数をアドレス値として read を行う，というループを繰り返すものである．マルチスレッドにした理由は，read 命令を発行したあと，その結果が返ってくるまでのブロッキング時間をマルチスレッド

により隠蔽出来る等の理由で高性能化が見込めるからである．結果は図 5 のとおりである．スレッド数が 1000 に近い領域では，性能が頭打ちになっていることが見受けられる．つまりスレッド数は十分で，この 4.8M IOPS 程度という数値が実験的なローカル入出力の最大性能値といえる．

図 5: スレッド数と Local Storage に対するランダムリードの性能

3.3.2 RDMA の性能測定 (RDMA 通信)

RDMA 通信の性能測定実験においてはコア固定を行っている．[1]

まず，様々なコネクション数において RDMA の性能測定を行った．コネクション数 n に対して 2n+1(2n 個の RDMA の送信・受信スレッドと 1 個のワークロードを管理するメインスレッド) が走っている．結果は図 6 のとおりである[2]．コネクション数の増加に伴って最初は OPS が増加しているが，11 から性能が低下しその後は停滞している．これは実験マシンの物理コアが 1 ソケットあたり 22 個であることに対し，11 コネクションのときには，23 個のスレッドがそれぞれコアに固定されることにより，スレッドがソケットを跨ぐ状況が発生しているためと考えられる．

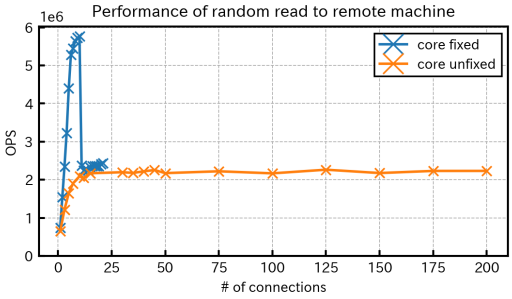

図 6: コネクション数とリモートマシンに対する RDMA 通信の性能

3.3.3 リモート入出力の性能測定 (動的入出力スレッド方式)

最後に，本論文の主題であるリモート入出力の性能測定実験を行った．[3]この節では動的に入出力スレッドを管理する方式の実験結果を述べる．

まずコネクション数を変化させて IOPS を測定した．RDMA 通信の時と同様に，コネクション数 n に対して 2n+1(2n 個の RDMA の送信・受信スレッドと 1 個のワークロードを管理するメインスレッド) が走っている．結果はコネクション数の増加に対して OPS は単調減少でありコネクション数 1 のときに，70K IOPS 程度となっている．

3.3.4 リモート入出力の性能測定 (静的入出力スレッド方式)

次に，静的入出力スレッドの測定を行った．ベースラインの実験は理論的上限値なので，比較のためグラフに破線として結果を再掲する．[4]

まずコネクション数を変化させてリモート入出力の性能を測定した．コネクションあたりの入出力スレッド数を 5 に固定した．結果は図 7 のとおりである．コネクション数が少ないところでは，リモート入出力の性能は，ローカル入出力の性能に律速されている．またコネクション数が多いところでは，RDMA 通信の性能に律速されており，2.3M IOPS を記録している．これが性能最大値である．

次に，入出力スレッド数を変化させてリモート入出力の性能を測定した．コネクション数は 10 に固定してある．結果は図 8 のとおりである．

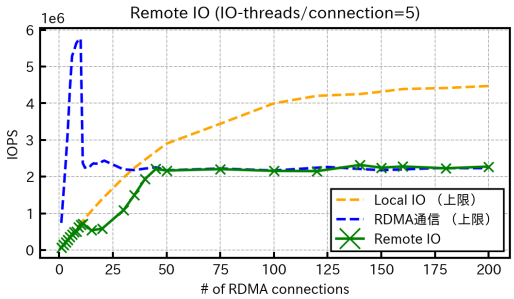

図 7: コネクション数とリモートストレージに対するリモート入出力（静的）の性能

3.3.5 ランダムリード実験のまとめ

ローカル入出力・RDMA 通信・リモート入出力（動的・静的）のリードサイズ別の最大値を図 9 に表す．ローカル入出力と RDMA 通信に関しては，偶然にも概ね同程度の性能値を出しているものの，その複合ワークロードであるリモート入出力は大きく性能が異なる．

リモート入出力（動的）では，著しく性能が低い一方，リモート入出力（静的）では，RDMA 通信やローカル入出力と同じオーダの性能を出している．リモート入出力の動的手法では，動的にスレッドを生成するため，スレッド生成のコストのオーバーヘッドがある．それにより静的手法より多少の性

(注1)：コア固定とは sched_setaffinity() によって，各スレッドを特定コアに張り付けることで，OS によるコア移動に伴う無駄なオーバーヘッドを取り除いたものである．
(注2)：物理コア数を超えるスレッド数は固定することができないので，コア固定のグラフは途中で途切れている．
(注3)：コア固定を行った場合と，行わない場合両方で計測したが，行わない場合のほうが常に性能が良かったため，コア固定を行わない場合のみ掲載する．
(注4)：実際には，パラメータとして様々なコネクション数と入出力スレッド数の組み合わせで実験を行ったが，ここでは紙面の都合上，どちらか片方を固定した結果のみ掲載する．

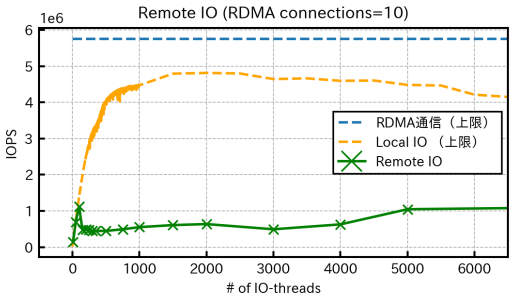

図 8: 入出力スレッド数とリモートストレージに対するリモート入出力（静的）の性能

能低下は認められるはずだが，今回の測定結果では 32 倍も性能が低下しており，スレッド生成のコストだけでは説明がつかない．

ここまで性能差が開く理由の現段階の仮説としては，動的手法では入出力スレッドを生成した後，生成した入出力スレッドに CPU 時間が中々渡らないため，入出力スレッドが全然実行されていないことが考えられる．これは，RDMA の受信のためのポーリングでは CPU 時間を要求する一方，read 命令は割込であり，ポーリングほど CPU 時間を要求しないため，OS スケジューラが適切に CPU 時間を配分できなくなっている状況により引き起こされているのではないかと理由付けをすることができる．

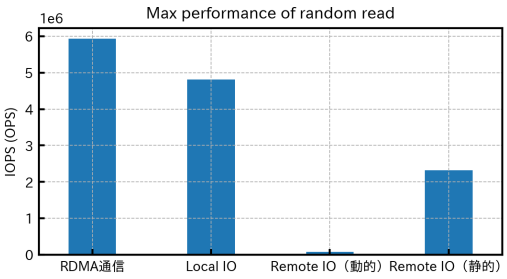

図 9: 512Byte ランダムリードの各方法での最大性能値

3.3.6 IO リプレイ

SQL クエリを実行したときの IO ログ（どのアドレスを読んだか）の履歴の通りに IO を発行すること（以下 IO リプレイと呼ぶ）で，擬似的にクエリの実行性能を測定することができる．OLAP 系ベンチマークである TPC-H のクエリ 3 を擬似実行することで，ランダムリードとは異なる実際のクエリの IO パターンでの性能測定も行った．なお用意した IO ログは 16KiB 単位でのアクセスだったので本実験では明記しない限り 16KiB アクセスを行っている．

まず入出力スレッド数を 10000 に固定してローカル入出力とリモート入出力のそれぞれで TPC-H クエリ 3 の IO リプレイを行った．結果は図 10 である．ローカル入出力のほうが性能が良いため当然実行時間は短いが，両者に劇的な差があるわけではない．

次にローカル・リモート入出力のそれぞれで入出力スレッド数を変化させて挙動を確かめた．比較のため 16KiB ランダムリード実験も参考として行った．ローカル入出力の結果は図 11 でリモート入出力は図 12 のとおりである．僅かではあるが，IO リプレイのほうが性能が良い領域がいくつか存在する．これは IO リプレイでは IO アクセスに局所性があり，キャッシュヒット率がランダムリードに比べて向上するためであるためと考えられる．

最後に参考として 512B の IO リプレイ実験も行った．上記の通り用意したログとはアクセスサイズが異なるので，これは完全なクエリの疑似実行ではないが，アクセスサイズが小さいときの挙動を確かめるために実験を行った．結果は図 13 のとおりである．入出力スレッドが少ない領域では，キャッシュヒット率の高さから IO リプレイのほうが性能が良いが，入出力スレッド数が増えてくるとスレッド管理コストがオーバヘッドとなりランダムリードを下回る性能となっている．これは実装上の問題と考えられるため今後改善が望まれる．

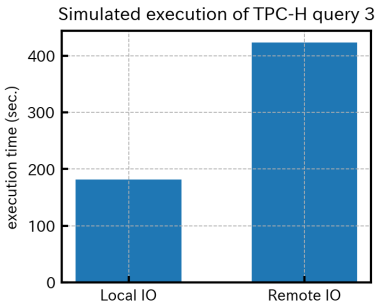

図 10: 16KiB での IO リプレイのローカル入出力とリモート入出力の実行時間の比較

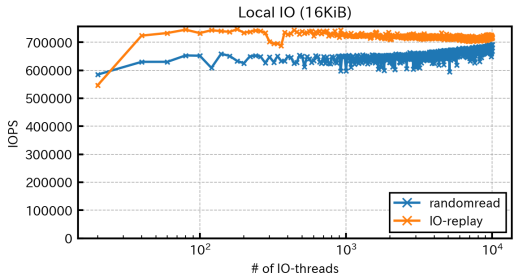

図 11: 16KiB でのランダムリードと IO リプレイのローカル入出力

4. 関 連 研 究

第 1. 節で述べたように，RDMA を用いたリモート入出力についての研究は現状なされていないが，RDMA を用いた通信（メモリ間通信）については，どのように DBMS に応用できるか数多くの研究がなされている．

Fent らは DBMS のサーバとクライアント間の通信に着目した [2]．その通信路上では，SQL クエリとその結果が転送されるが，ここに RDMA を導入し大幅なパフォーマンス向上に成功している．Fent らが RDMA を DBMS に導入するにあたっ

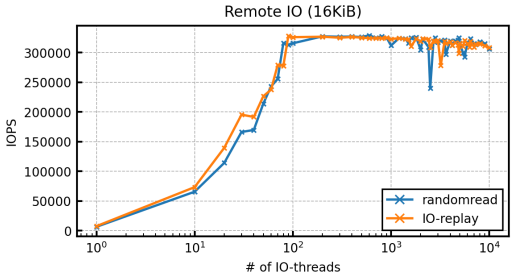

図 12: 16KiB でのランダムリードと IO リプレイのリモート入出力

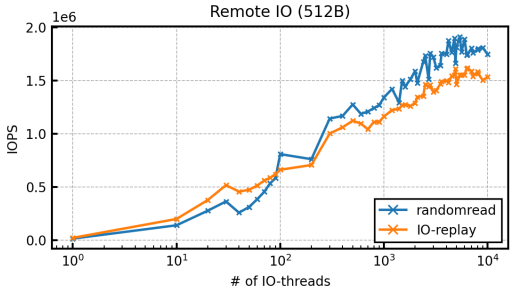

図 13: 512B でのランダムリードと IO リプレイのリモート入出力

て示した効率的なデータ構造は本研究のように入出力まで行う際も参考にすることが出来る．

また，RDMA を採用した分散 DBMS ではノード間通信が高速低遅延になり，アルゴリズムの前提条件が崩れるということがしばしばある．これに対し，ノード間通信を極力減らそうとする従来のアルゴリズムではなく，新たなアルゴリズムを考えて RDMA の力を最大限使い切るという研究がなされている [4]〜[8]．例えば [21] の研究では，従来最優先事項であったノード間通信を避けるということを辞め，データレコードに対して競合が発生する時間を最小化する Chiller というアルゴリズムを提案することで，パフォーマンスが 2 倍向上することを示した．これらの研究で対象とされているのはマシン間通信でありメモリ上で完結する故に，ストレージまで入出力を発行する本研究には直接活かすことは出来ない．しかし，本研究でリモート入出力のコストが下がった結果，今後これらの研究のように変化したパワーバランスの上でアルゴリズムを再設計することで，さらなる発展が望めると考えられる．

5. おわりに

本研究により，RDMA を使用したリモート入出力を行う際の最適な使用方法とその最大性能値が明らかになった．このリモート入出力は並列 DBMS を構成するマシン間で通信する際に用いることを主に想定している．並列 DBMS において，本研究の RDMA を用いたリモート入出力を使用することで，高速な Disaggreated storage 型のストレージアーキテクチャを実現することが出来るようになった．また本研究により得られた知見は DBMS 以外にも，リモート入出力を行うアプリケーションであればどのようなものでも適用することができ，その貢献は大きい．

文　献

[1] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden, “Speedy transactions in multicore in-memory databases,” ••, pp.18–32, 2013.

[2] P. Fent, A. vanRenen, A. Kipf, V. Leis, T. Neumann, A. Kemper, and F.-S.-U. Jena, “Low-latency communication for fast dbms using rdma and shared memory,” 2020. https://github.com/pfent/L5RDMA

[3] S. Babu and H. Herodotou, “Massively parallel databases and mapreduce systems,” Foundations and Trends® in Databases, pp.••–••, 2013.

[4] D.Y. Yoon, M. Chowdhury, and B. Mozafari, “Distributed lock management with rdma: Decentralization without starvation,” ••, pp.1571–1586, Association for Computing Machinery, May 2018.

[5] G. Chatzopoulos, A. Dragojević, and R. Guerraoui, “Spade: Tuning scale-out oltp on modern rdma clusters,” ••, pp.80–93, Association for Computing Machinery, Inc, Nov. 2018.

[6] C. Barthels, IngoMu ¨ ller, K. Taranov, G. Alonso, T. Hoefler, “Strong consistency is not hard to get: Twophase locking and twophase commit on thousands of cores,” ••, 第 12 巻, pp.2325–2338, VLDB Endowment, 2020.

[7] C. Binnig, A. Crotty, A. Galakatos, T. Kraska, and E. Zamanian, “The end of slow networks: It’s time for a redesign,” 2016.

[8] E. Zamanian, C. Binnig, T. Harris, and T. Kraska, “The end of a myth: Distributed transactions can scale,” 2017.

[9] E. Zamanian, X. Yu, M. Stonebraker, and T. Kraska, “Rethinking database high availability with rdma networks,” ••, vol.12, pp.1637–1650, VLDB Endowment, 2018.

[10] Q. Cai, W. Guo, H. Zhang, D. Agrawal, G. Chenz, B.C. Ooi, K.L. Tan, Y.M. Teo, and S. Wang, “Efficient distributed memory management with rdma and caching,” ••, vol.11, pp.1604–1617, Association for Computing Machinery, 2018.

[11] B. Li, Z. Ruan, W. Xiao, Y. Lu, Y. Xiong, A. Putnam, E. Chen, and L. Zhang, “Kv-direct: High-performance in-memory key-value store with programmable nic,” ••, pp.137–152, Association for Computing Machinery, Inc, Oct. 2017.

[12] T. Ziegler, S.T. Vani, C. Binnig, R. Fonseca, and T. Kraska, “Designing distributed tree-based index structures for fast rdma-capable networks,” ••, pp.741–758, Association for Computing Machinery, June 2019.

[13] X. Wei, Z. Dong, R. Chen, H. Chen, and S.J. Tong, Deconstructing RDMA-enabled Distributed Transactions: Hybrid is Better!, ••, 2018. www.usenix.org/conference/osdi18/presentation/wei

[14] C. Wang, K. Huang, and X. Qian, “Comprehensive framework of rdma-enabled concurrency control protocols,” ••, pp.••–••, Feb. 2020. http://arxiv.org/abs/2002.12664

[15] A. Kalia, M. Kaminsky, and D.G. Andersen, “Using rdma efficiently for key-value services,” ••, vol.44, pp.295–306, Association for Computing Machinery, Feb. 2015.

[16] C. Mitchell, Y. Geng, J. Li, and N.Y. University, “Using one-sided rdma reads to build a fast, cpu-efficient key-value store,” 2013.

[17] A. Dragojević, D. Narayanan, E.B. Nightingale, M. Renzelmann, A. Shamis, A. Badam, and M. Castro, “No compromises: distributed transactions with consistency, availability, and performance,” Proceedings of the 25th symposium on operating systems principles, pp.54–70, 2015.

[18] Y. Chen, X. Wei, J. Shi, R. Chen, and H. Chen, “Fast and general distributed transactions using rdma and htm,” Proceedings of the Eleventh European Conference on Computer Systems, pp.1–17, 2016.

[19] A. Kalia, M. Kaminsky, and D.G. Andersen, FaSST: Fast, Scalable and Simple Distributed Transactions with Two-sided (RDMA) Datagram RPCs, USENIX Association, 2016.

[20] K. Goda and M. Kitsuregawa, “The history of storage systems,” Proceedings of the IEEE, vol.100, no.Special Centennial Issue, pp.1433–1440, 2012.

[21] E. Zamanian, J. Shun, C. Binnig, and T. Kraska, “Chiller: Contention-centric transaction execution and data partitioning for modern networks,” ••, pp.511–526, Association for Computing Machinery, June 2020.