DEIM2021 B25-3

# RDMA を用いたリモート入出力性能の実験的考察

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 を構成する各マシン間の通信に，RDMA (Remote Direct Memory Access) と呼ばれるリモートメモリアクセス技術を適用することを試みる研究が近年見られる．それらの研究では，RDMA 自体の性能については測定・評価しているが，それらはリモートマシンのメモリにアクセスするのみに留まり、その先のストレージにアクセスすることは少ない。故に RDMA を経由してリモートマシンのストレージに対して読み書きを行う，より包括的なリモート入出力については，その性能や特性が明らかになっていない．本論文ではこのリモート入出力について，実験的に性能を測定することでその動作特性を明らかにし，並列 DBMS においてどのようにリモート入出力を発行・処理すればよいか考察を行った．その結果，静的に入出力スレッドを管理する方法では，2.3M IOPS を記録し，動的に管理するものよりも約 32 倍性能が高いことが発見された．

**キーワード**　並列 DBMS，入出力，ネットワーク，RDMA，InfiniBand

## 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，近年 DBMS のネットワーク以外のコンポーネントが性能向上を遂げていることも相まって，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].

そこで近年，ネットワークがボトルネックであることを解消すべく，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 DBMS に導入しようという研究動向が存在する [4–8]．これは単に TCP/IP を RDMA で置き換えれば済む問題ではなく，上記の通り，既存の DBMS の多くは TCP/IP を暗黙のうちに仮定して設計されているので，RDMA の導入にともなってアーキテクチャやシステムソフトウェアの再設計を行わないと，RDMA の性能を最大限引き出せなかったり，そもそも使用できなかったりする．

RDMA の導入には様々なアプローチから研究がなされている．RDMA は TCP/IP とは異なり，本来通信というよりメモリアクセスの技術であるので，適切に抽象化して DBMS で使用する必要がある．そこで，どのように抽象化すればよいかという研究が盛んになされている [2,9–12]．また RDMA が提供する様々な通信方式のうちどれが DBMS に適しているかを探る研究もなされている [13–19]

しかし，これらの研究では RDMA 自体の性能や，あるいはシステム全体での性能評価は行われている一方で，RDMA を経由してリモートマシンのストレージに対して入出力を発行する，より包括的な経路における性能については測定されていない．つまり，RDMA を導入した際，自マシンのメモリとリモートマシンのメモリ間の通信については，マイクロベンチマークのような基礎的な性能評価も行われているが，自マシンのメモリとリモートマシンのストレージ間の通信については，その性能は未知であるということである．

本研究では，この RDMA を使用したリモート入出力についての性能測定及び最適化を行う．リモート入出力は機能的にはシンプルであるが，性能に配慮しながらそれを実現するには，RDMA が提供する様々な通信方式の選定や，メッセージのデータ構造の考案，メッセージングを行うスレッド数やブロックサイズの調整など考慮すべきことが多くある．本研究ではこれらを同時に包括的に比較することで，リモート入出力の最適実装やその実験的最大性能値を明らかにする．

この実験により，RDMA を使用した高性能なリモート入出力の方法が示されれば，RDMA を用いた並列 DBMS において他ノードのストレージにアクセスするコストが低くなり，大きなブレイクスルーが起こる可能性がある．また DBMS 以外でもリモート入出力を行うシステムであれば，本研究の知見が生かされると考えられ，その貢献は大きい．

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・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，Fig. 1 のように Shared Disk，Shared Nothing の 2 つのアーキテクチャに大別することができる．

Shared Disk 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．ただし，必ずしも物理的に Fig. 1(a) のような構成になっている必要はなく，各マシンに専用のストレージが接続されているような場合でも，ネットワーク経由など何らかの形でリモートマシンのストレージにもアクセスする方法が提供されていれば，Shared Disk アーキテクチャとなる．

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．

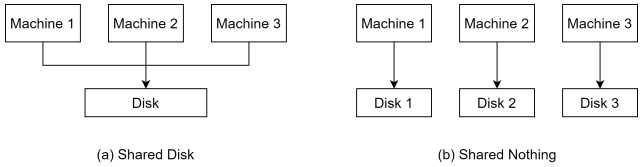


Fig. 1: ストレージアーキテクチャの分類

本研究が想定するアーキテクチャは Shared Disk である．物理的に各マシンが専用のストレージを持つ並列 DBMS において，ネットワーク経由で RDMA を介したリモートストレージへの入出力を行うことで，各マシンは任意のストレージにアクセスすることができるからである．DBMS の標準的なネットワークスタックである TCP/IP を使用してこのような処理を行うと，TCP/IP は遅いためネットワークがボトルネックになってしまい性能が伸びないことが従来の問題点であったが，本研究によって，RDMA の性能を最大限発揮するシステムソフトウェアやリモート入出力の実験的性能上限が明らかになれば，並列 DBMS におけるブレイクスルーとなりうる．

### 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．

RDMA が低遅延な通信を提供できる理由として，ゼロコピー通信の寄与が大きい．TCP/IP では，通常ユーザメモリ空間上にあるデータは，OS のデータバッファに CPU を使用してコピーされた後に，リモートマシンへと送信される．しかし，ゼロコピー通信では，ユーザメモリ空間上にあるデータは CPU やカーネルの関与なしで，RDMA NIC (RNIC) によって直接リモートマシンへと送信されるので，遅延が非常に少ない．

RDMA はそのインターフェースとして verbs API [20] を提供している．verbs と呼ばれるプリミティブを RNIC が管理するキューに入れることで，制御が可能となる．この verbs には one-sided プリミティブ (e.g. READ，WRITE， ATOMIC) と two-sided プリミティブ (SEND，RECEIVE) の二種類がある．どちらも同じような機能 (i.e. 送信・受信) を提供しているが，その実装や性能には差がある．one-sided は一般により高いパフォーマンスであり，また，リモートマシンの CPU 関与すら必要としない．一方，two-sided はリモートマシンの CPU 関与は必要とするが，one-sided よりは通信を抽象化しており，シンプルで使いやすい．つまり，ただでさえ原始的なプロトコルである RDMA であるが，one-sided のほうがより原始的で低レイヤーのインターフェースであると言える．

またプリミティブという分類以外にも，IP における TCP/UDP のように RDMA にもサービスタイプという分類があり，Reliable な Connection を提供する RC や Unreliable な Datagram を提供する UD などがある．

### 2.3 RDMA を用いたリモート入出力

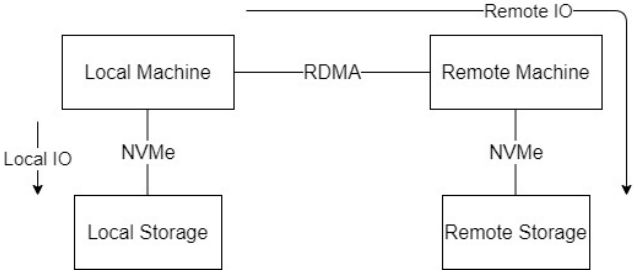


Fig. 2: RDMA を用いたリモート入出力を行うアーキテクチャ

本研究におけるリモート入出力の概要を説明する．Fig. 2 のように，2 台のマシンがあったときに，片方からもう片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．Fig. 1(a) のようにマシンが 3 台以上存在するときも，任意のマシン間をネットワーク接続することにより，結局 Fig. 2 のような 2 マシン間のリモート入出力の状況に帰着させることができる．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．つまり自マシンのストレージにアクセスする際の手段は，read() のような通常のローカルストレージへのアクセスと同じ手段を用い，そ

の命令をリモートマシンからトリガーするのに，RDMA を使用する．

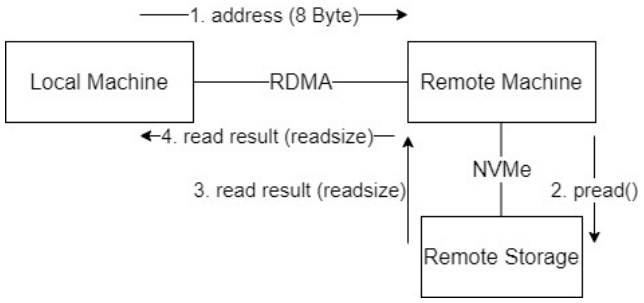


Fig. 3: リモート入出力のフローと，各フローにおけるメッセージ内容とそのサイズ

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 Fig. 3 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 に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．

このようなリモート入出力の系を考えると，その性能を最大化するためには，各マシンで入出力を行うスレッド数，RDMA を用いて張るコネクション数，アクセスの際のリードサイズの大きさなど考慮すべき事柄が多くある．本研究の実験では，これらパラメータの様々な組み合わせに対して性能評価を行うことで，最適な RDMA の使用方法を示すとともに，その最大性能も明らかにする．

### 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，リモート入出力の性能がローカル入出力（つまり通常の入出力）の性能を上回ることは考えられない一方で，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 Fig. 3 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，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（つまりローカルマシンとリモートマシンで計 4 つのスレッドが存在することになる）．コネクションは一つしか張らなくてもランダムリード自体は行える上に，上記のように多重化もしているのでパフォーマンスも著しく悪いわけではない．しかし，コネクションが一つでは RDMA やストレージアクセスがボトルネックになるのではなく，CPU などがボトルネックになり十分にハードウェアの性能を発揮できない可能性があるため，コネクションを複数張り，ランダムリードを行うスレッドを複数用意することで，性能向上を図っている．

### 3.2 実験環境

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．

Table 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) |

両マシンの仕様は同一である．詳しい環境を Table 1 に示す．

InfiniBand 上で RDMA を使用する際は，libibverbs や librdmacm のようなライブラリが提供する API を使用する．これら 2 つのライブラリの違いとしては，librdmacm は libibverbs のラッパーであり，より簡易なインタフェースを提供している．本実験では最適化を行うため，より低レイヤであり直接 API を使用できる libibverbs を用いて実装を行う．

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 [21]．SSD にアクセスする方法として，ストレージに紐づく 10 個のデバイスファイルを O_DIRECT フラグを有効にしてオープンし，それらにラウンドロビン方式で pread 命令を発行する．ここで O_DIRECT フラグとは OS によるキャッシングを防ぐもので，より正確な入出力の性能を測定することができる．

### 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，リードサイズを独立変数として設定して実験した．1000 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 Fig. 4 のとおりである．IOPS を高めるにはリードサイズを小さくすればよいことがわかる．

また，リードサイズが 512B のときに，スレッド数は 1000 で十分か確かめるために，今度はリードサイズを 512B に固定し，スレッド数を変化させて実験を行った．結果は Fig. 5 のとおりである．スレッド数が多ければ多いほど read 性能は高くなっているが，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．

#### 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．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．

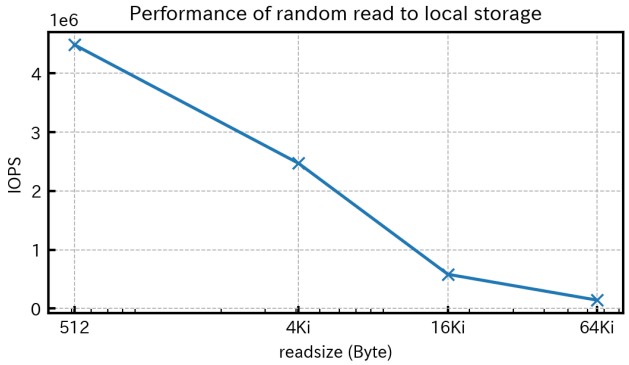


Fig. 4: リードサイズと Local Storage に対するランダムリードの性能

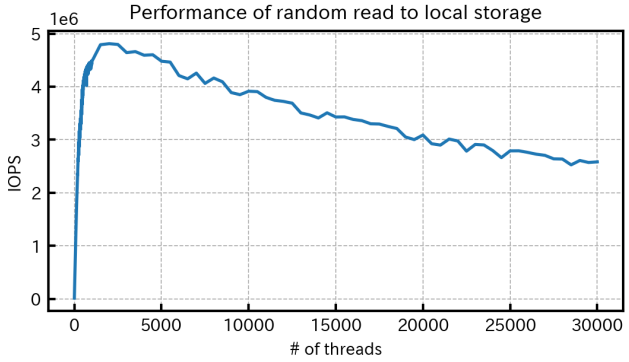


Fig. 5: スレッド数と Local Storage に対するランダムリードの性能

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．リードサイズは 512Byte で固定である．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 Fig. 6 のとおりである [1]．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．コネクション数が 10 のときに，OPS は最大値である 6M 弱に達しており InfiniBand が提供する RDMA が非常に高速なことがわかる．

次に，リードサイズを変化させて RDMA の性能測定を行った．先程の実験からコネクション数が 10 において性能が最大化されると明らかになったので，コネクション数は常に 10 としている．結果は Fig. 7 のとおりである．リードサイズが大きくなるにつれて，RDMA によるデータ転送に時間がかかるようになるため，OPS は減少している．

1：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．

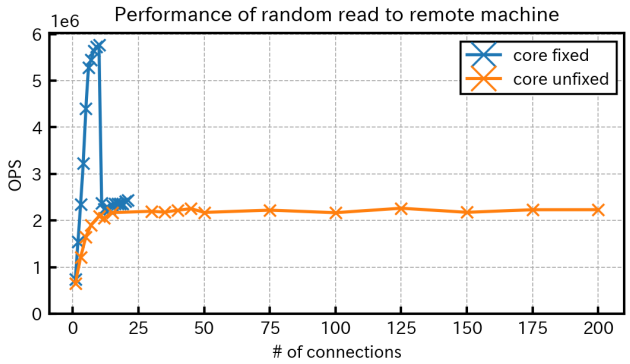


Fig. 6: コネクション数とリモートマシンに対する RDMA 通信の性能

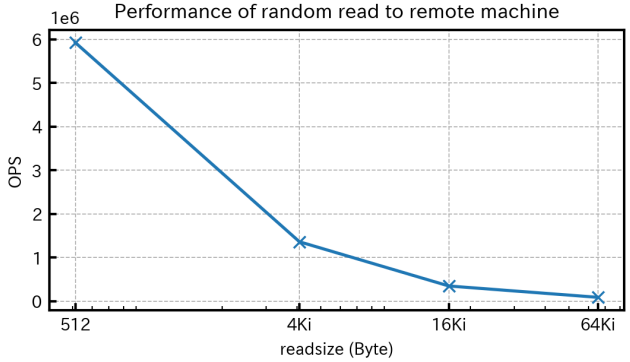


Fig. 7: リードサイズとリモートマシンに対する RDMA 通信の性能

### 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．[2]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．リードサイズは 512Byte で固定である．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 Fig. 8 のとおりである．コネクション数は少ないほど性能が良くなっており，コネクション数 1 のときに，70K IOPS 程度となっている．この理由は，ボトルネックがリモートストレージに対する入出力にあるため，コネクションを増やして RDMA 用のスレッドを無駄に増やすよりも，入出力を最大限行えるよう可能な限りリモートマシンで走る RDMA のためのスレッド数を減らしておいたほうが良いからだと考えられる．

次に，リードサイズを変化させてリモート入出力の性能測定を行った．先程の実験からコネクション数が 1 において性能が最大化されると明らかになったので，コネクション数は常に 1 としている．結果は Fig. 9 のとおりである．リードサイズが大きくなるにつれて，ストレージに対する入出力や RDMA によるデータ転送に時間がかかるようになるため，IOPS は減少している．

2：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．

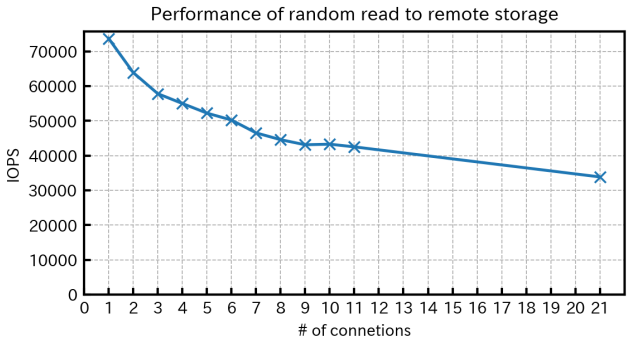


Fig. 8: コネクション数とリモートストレージに対するリモート入出力（動的）の性能

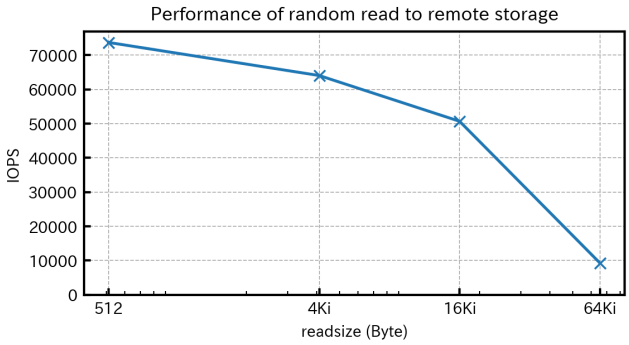


Fig. 9: リードサイズとリモートストレージに対するリモート入出力（動的）の性能

### 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．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．

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．リードサイズは 512Byte で固定である．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 Fig. 10 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．やはりリードサイズは 512B で固定である．コネクション数は 10 に固定してある．結果は Fig. 11 のとおりである．入出力スレッドが小さい領域では，ローカル入出力がボトルネックになっている．一方で，それ以外の領域では，ローカル入出力と RDMA 通信のどちらにも達していない．

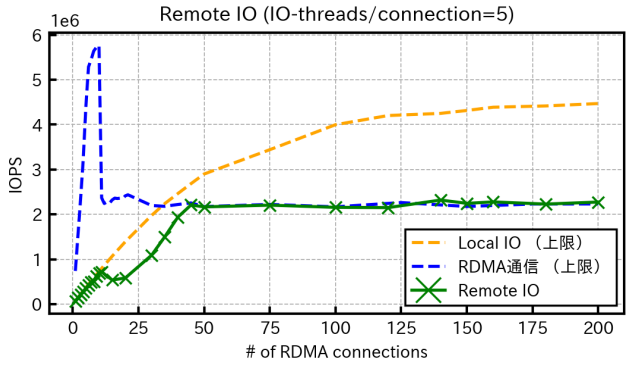


Fig. 10: コネクション数とリモートストレージに対するリモート入出力（静的）の性能

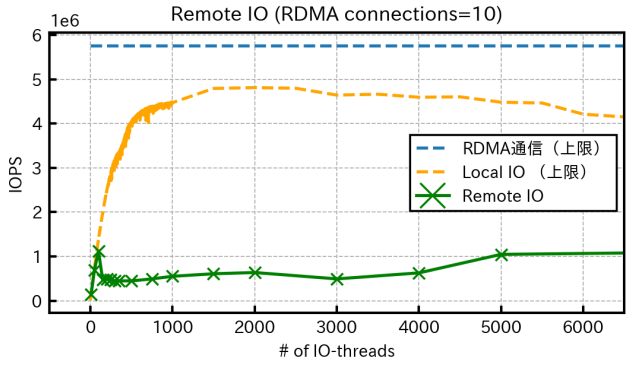


Fig. 11: 入出力スレッド数とリモートストレージに対するリモート入出力（静的）の性能

#### 3.3.5 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 Fig. 12 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．

RDMA を利用したリモート入出力としては，静的手法が断然優れておりこちらを採用すれば良いというのが本研究の結論

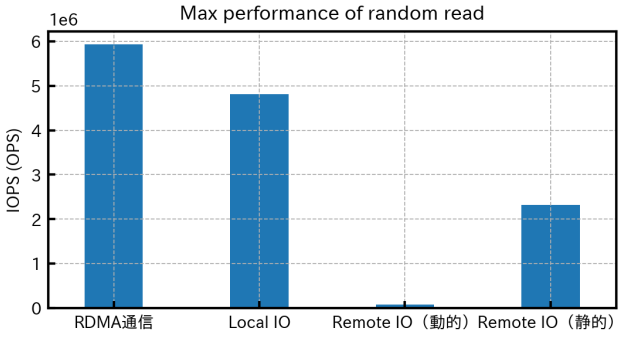


Fig. 12: リードサイズ別の各ワークロードの最大性能値

であるが，動的手法の性能の低さの原因も，明らかにするべきであると考えられる．

## 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．

例えば分散 DBMS を構成するマシン間で RDMA 通信する際（トランザクション処理の際）に，RDMA が提供する 2 つのプリミティブのうち，one-sided プリミティブと two-sided プリミティブのどちらを使用すればよいかという点で活発な議論がなされている．FaRM [17] という手法では，主に one-sided のみを使用しており，DrTM+R [18] では，完全に one-sided のみを使用している．一方，FaSST [19] という手法では，two-sided のみを使用している．さらに DrTM+H [13] では，分散トランザクション処理のアルゴリズムに Optimistic Concurrency Control(OCC) を採用し，OCC が 4 つのフェーズから成立つことに注目し，各フェーズで行われる通信の特性を考慮した上で，2 つのプリミティブをハイブリッドに使い分けるという手法をとっている．本研究では，ランダムリードという基礎的なマイクロベンチマークの側面が強いワークロードであったため，このような最適化の出番はなかったが，実際に特定の並列 DBMS に RDMA のリモート入出力を組み込む際は，DrTM+H [13] のような，通信内容まで考慮した最適化をかけることで，さらなる性能向上が期待できる．

また，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，従来ほどコストがかからなくなったが故に，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 [22] の研究では，従来のアルゴリズムの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，従来のノード間通信を避けるアルゴリズムよりも，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，相対的に他のシステムコンポーネントとのパワーバランスが変化したことにより，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 Shared Disk 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．

今後の課題として，リモートマシン上のプログラムロジックの改善により，リモート入出力の性能を更に向上させることがあげられる．リモート入出力は，ローカル入出力とネットワークに論理的に分解することができるが，理想的には，両者のうち性能が悪い方が理論上のリモート入出力の性能最大値である．本研究では，この最大性能値にはまだ達していないので，リモートマシン上のプログラムロジックの改善を最優先の課題としたい．

また，本研究ではランダムリードをワークロードとして想定したが，ランダムライトやシークエンシャルな読み書きについては実験出来ていない．実際のデータベース負荷においてはこれらの入出力も発生し得るので，本研究と同様に実験的考察がなされることが求められる．また，本研究では様々な測定条件で試行することにより最適な RDMA の使用方法を探索したが，本研究では考えなかった未探索の測定条件もある．例えば RDMA を使用する際には，非常に多くの API が提供されているが，本研究ではそのうち代表的なものしか使用していない．また，RDMA で転送するメッセージのデータ構造は開発者が任意に決めることが出来，より効率的な実装方法が存在する可能性もある．今後，このような測定条件についてもさらなる探索がなされることも期待される．

## 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. van Renen, 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.

[3] S. Babu and H. Herodotou, “Massively parallel databases and mapreduce systems,” *Foundations and Trends® in Databases*, 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, 5 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, 11 2018.

[6] C. Barthels, I. Mu¨ller, K. Taranov, G. Alonso, and T. Hoefler, “Strong consistency is not hard to get: Twophase locking and twophase commit on thousands of cores,” vol. 12, 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, 10 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, 6 2019.

[13] X. Wei, Z. Dong, R. Chen, H. Chen, and S. J. Tong, *Deconstructing RDMA-enabled Distributed Transactions: Hybrid is Better!* 2018.

[14] C. Wang, K. Huang, and X. Qian, “Comprehensive framework of rdma-enabled concurrency control protocols,” 2 2020.

[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, 2 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,” in *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,” in *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] G. Kerr, “Dissecting a small infiniband application using the verbs api,” *arXiv preprint arXiv:1105.1827*, 2011.

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

[22] 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, 6 2020.