DEIM2022 K23-3

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測定と問合せ処理への影響

加藤　滉貴† 　小沢　健史† 　合田　和生† 　喜連川　優†,††

† 東京大学　〒 113–8656 東京都文京区本郷 7-3-1
†† 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　近年，高速ネットワークによって結合された計算機上で二次記憶を分散して管理することを特徴とする「Disaggregated storage architecture」が注目されている．サーバに対するデータの局所性が存在するので，当該ストレージアーキテクチャ上に高性能なデータベースシステムを構築するためには，データの距離に応じてアクセスコストが異なることに起因する，アフィニティ問題を解決する必要がある．本研究では，当該ストレージアーキテクチャに基づいて構成された並列データベースを対象として，問合せ処理の効率を向上するための，アフィニティを考慮したデータと演算の配置方法・入出力の発行方法を検討し，実装を用いた実験によりその有効性を示した．

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

1　はじめに

近年，InfiniBand や 100Gb Ethernet など高速ネットワークの技術がコンバージドネットワーク [1] として採用され，データベースを構成する際に使用されることがある．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 [2–18].

歴史的には，データベースシステムを構成する際，ネットワークに採用される技術としては TCP/IP がデファクトスタンダードである．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データベースシステム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データベースシステムにおいてボトルネックとなることが多い [19].

さて，大規模データを高速にデータベースシステムで処理するための手段として，並列データベースシステムを構成して，負荷を複数のサーバで分散するという手法がある．このためのストレージアーキテクチャとして，Shared-nothing [20] と Shared-storage [21] 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，ネットワークが高速化した結果，Disaggregated storage architecture（非集約型ストレージアーキテクチャ）という新しいストレージアーキテクチャ(Fig. 1参照) が考案されている [22, 23].

Disaggregated storage architecture（以下，Disaggregated 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．

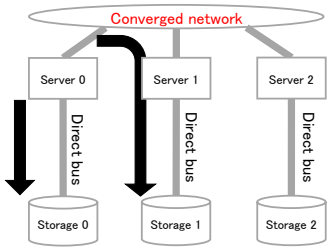

Fig. 1: Disaggregated storage

そこで，本研究ではコンバージドネットワークを用いた Disaggregated storage 型の並列データベースシステム上で，どのようにオペレータ，IO，データのアフィニティを調整したら最大性能を引き出せるかを検討した．

2　ストレージアーキテクチャと並列データベースシステム

2.1　並列データベースのストレージアーキテクチャ

並列データベースをストレージアーキテクチャの視点から見ると，ストレージをサーバ間で共有するか否かという観点で，Fig. 2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 [20, 21].

Shared-storage アーキテクチャでは，複数のサーバ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，各サーバはリモートサーバ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．

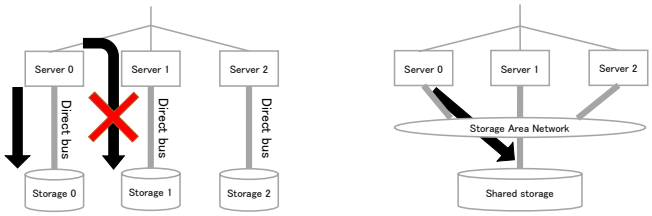

Fig. 2: 既存のストレージアーキテクチャ

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 Disaggregated storage が存在する（Fig. 1）．Disaggregated storage は，Shared-nothing と Shared-storage の特徴を併せ持つハイブリッドなアーキテクチャである．Shared-nothing のように専有のストレージを持ち，そのストレージへは高速でアクセスすることができる．また他サーバのストレージにもコンバージドネットワーク経由でアクセスできるため，Shared-storage のようにストレージ空間はグローバルである．このように Disaggregated storage は両者の利点を併有する．

一方で Disaggregated storage に於いて，サーバは自ストレージへと他ストレージでアクセスコストが異なる．そのため，潜在的には非常に高いパフォーマンスを持つストレージアーキテクチャではあるが，アフィニティを上手に調整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問題が存在する．そしてこの問題は現在未解決である．そこで本研究では，他ストレージへの高速なアクセス方法であるリモート IO を提案し実証した．

2.2 Remote Direct Memory Access (RDMA)

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 RDMA を採用する．Remote Direct Memory Access，通称 RDMA とはその名の通り，リモートサーバに存在するメモリ上に読み書きを行う技術である [5, 12, 18]．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．

3 Disaggregated storage architecture に於ける効率的なクエリ処理方式

3.1 RDMA を用いたリモート IO

本研究で提案するリモート IO の概要を説明する．2 台のサーバがあったときに，片方のサーバからもう一方サーバに接続された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 IO の想定である．

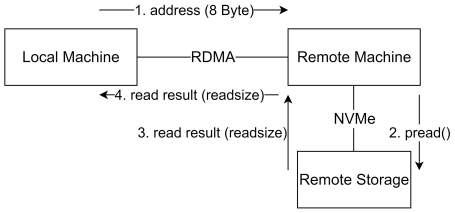

Fig. 3: リモート IO のフロー

リモート IO における具体的なメッセージ内容は Fig. 3のとおりである．ローカルサーバにおいて，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 を用いてリモートサーバに送信する．アドレス値を受けとったリモートサーバのプログラムは，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 でローカルサーバに送信することで，ローカルサーバは Remote Storage のデータを取得できリモート IO が完了する．なおこの一連のフローは，RDMA で通信する際に張るコネクションやリモートサーバが用意する IO スレッドを多重化することにより，フロー自体も性能のため多重化する．

3.2 Disaggregated storage 上でのクエリ処理

データベースでクエリ処理をする際は，IO を発行してページ単位でデータを取得した後，ページをパースしてレコードに分解する．レコードから必要なアトリビュートのデータを取得し，適宜演算を行う．

Disaggregated storage でクエリ処理するケースを考えると，データをどのサーバ（i.e. ローカルサーバ，リモートサーバ）でオペレータの演算を行うかという自由度が存在する．そこでこれらの状況を整理し，3 つの方式を検討した．

3.2.1 ローカルクエリ

データが存在するストレージに直接接続されているサーバで演算を行う．つまりローカルストレージのデータをローカルサーバで処理する．コンバージドネットワークを使用しないので，最もシンプルな方式であると言える．RDMA によるデータ転送をする必要がないので，低コストでパフォーマンスが良いことが期待される．フローを Fig. 4に示す．

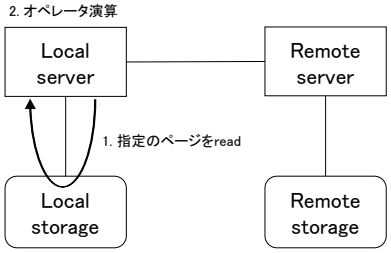

Fig. 4: ローカルクエリのフロー

3.2.2 リモートクエリ（ローカル処理）

リモート IO によって，ページをリモートストレージからローカルサーバへと転送したあと，ローカルサーバで演算を行う．フローを Fig. 5に示す．

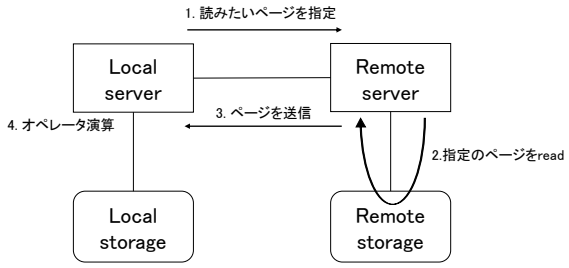

Fig. 5: リモートクエリ（ローカル処理）のフロー

3.2.3 リモートクエリ（リモート処理）

リモートストレージのページをローカルサーバへと転送する際，途中のリモートサーバで演算を行う．演算が SQL の where 句のような処理の場合，演算結果のレコード数は演算前のものより減少する場合がある．そのような場合，リモートクエリ（リモート処理）方式では，演算結果の（減少した）レコードのみを RDMA によってローカルサーバに送り返す．

リモートサーバからローカルサーバへのデータ転送量がリモートクエリ（ローカル処理）と比較して減少するので，効率的な方式であることが期待される．フローを Fig. 6に示す．

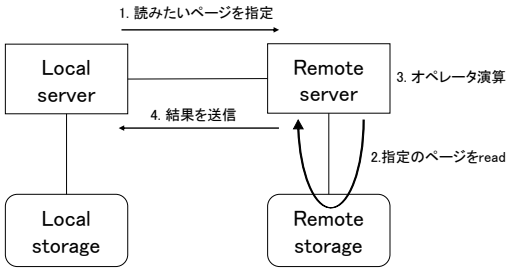

Fig. 6: リモートクエリ（リモート処理）のフロー

4 評価実験

4.1 実験概要

本研究で行う実験は大別すると 2 つにわけられる．一つ目は Disaggregated storage 環境下で IO を効率良く行う方法を明らかにする実験である．もう一つはオペレータと IO を組み合わせたクエリ処理を効率良く行う方式を明らかにする実験である．明記しない限り，IO に於けるブロックサイズは 4KiB である．

4.1.1 IO マイクロベンチマーク

リード要求を短期間に大量に発行する実験を行う．ワークロードとしてはランダムリードや第 4.4節で説明する IO リプレイである．リードサイズは固定長である．

リモート IO は，Fig. 7のように，ローカル IO と RDMA 通信部に分解することが出来る．リモート IO はこれらをブリッジングしたものであるので，性能最大値は 2 つのうち性能が低い方に律速される．そこでローカル IO と RDMA 通信をそれぞれ独立にマイクロベンチマークとして性能測定することで，リモート IO の理論的性能最大値を先に求めた．

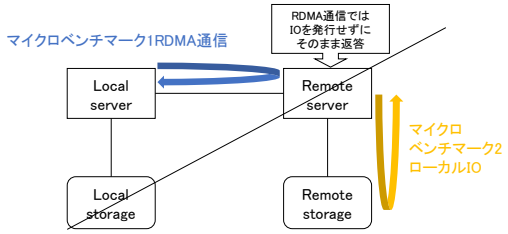

Fig. 7: リモート IO を 2 つに分割したマイクロベンチマーク

4.1.2 IO リプレイによるクエリ実行

ローカル IO，リモート IO の上にオペレータを組み合わせたワークロードを行う．具体的には IO を発行して取得したデータ（レコード）を SQL の where 句に相当する演算で処理することを行う．つまり擬似的にクエリ処理を行う．where 句の演算は CPU を主に消費するので，これにより IO のときよりもサーバの CPU 資源が消費されることになる．このような状況下で性能測定を行うことで，Disaggregated storage 上に構成された並列データベースに於いて，クエリ処理の性能特性を調査し，高速なクエリ処理の方式を実証する．

本ワークロードではローカル IO，リモート IO を用いてページ単位でストレージにアクセスする．ページにはデータベースのレコードが複数並んでおり，ページをパースしレコードを逐次 where 句にかけることでクエリを実行する．

4.2 実験環境

本研究で使用した実験環境について示す．実験に使用したプログラムは全て C 言語で記述した．

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)
OS のページサイズ	4KiB

本研究の実験環境は 2 台のサーバから構成される．これらのサーバは 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．両サーバの仕様は同一である．詳しい環境を Table 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 [24].

4.3 IO マイクロベンチマーク（ランダムリード）

まず最初に，ワークロードとしてランダムリードを採用し，RDMA 通信（第 4.3.1節参照），ローカル IO（第 4.3.2節参照），リモート IO（第 4.3.3節参照）の各方式で実験的考察を行った．リモート IO が本実験の主目的である．

4.3.1 RDMA 通信の性能測定

リモート IO では，リモートサーバは RDMA によってリード要求のアドレス値を受信し，そのまま IO へと中継する．しかし本マイクロベンチマークでは IO を発行せず，IO を発行したものと仮定してダミーデータを即座に送りかえす．これにより RDMA 側の通信性能を知ることが出来る．

マルチスレッド化し，複数の QP と呼ばれるコネクションを張り，それらで同時多重的にランダムリードを行った．結果は Fig. 8である．コネクション数によって，特に性能は変化していない．

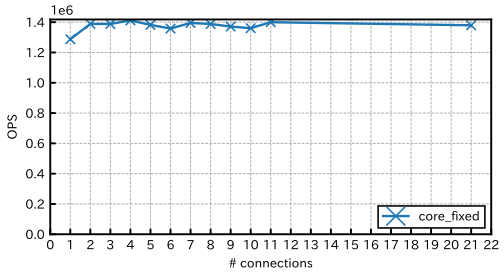

Fig. 8: 4KiB RDMA 通信

4.3.2 ローカル IO の性能測定

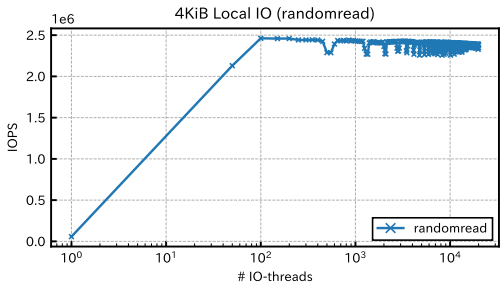

Fig. 9: 4KiB ローカル IO

リモート IO を構成する要素であるローカル IO に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サーバは 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．ローカル IO はリモート IO に対するベースラインという位置付けもある．

結果は Fig. 9である．スレッド数が大体 100 付近で性能が頭打ちしていることがわかる．

4.3.3 リモート IO の性能測定

RDMA のコネクション数とその各コネクションが生成する IO スレッド数の様々な組み合わせのパラメータで，リモート IO のランダムリードを行った．まずはリードサイズ 512B で，どちらか一方のパラメータを固定して，もう片方のパラメータを振って実験した．

RDMA のコネクション数を 10 に固定した結果は Fig. 10である．系全体の IO スレッド数を横軸とした [1]．参考としてリモート IO を構成する要素であるローカル IO と RDMA 通信の結果も掲載している．原理上これらよりリモート IO の性能が上回ることはない．ピークでも理論的性能最大値（この場合ローカル IO の性能値）との間には大きくギャップがある．

逆に，コネクション当たりの IO スレッド数を固定した結果は Fig. 11である．コネクション数を増やせば増やすほど性能が向上しており，コネクション数 100 付近では，ほぼ理論的性能最大値に達しており，理想的な性能に達していることがわかる．

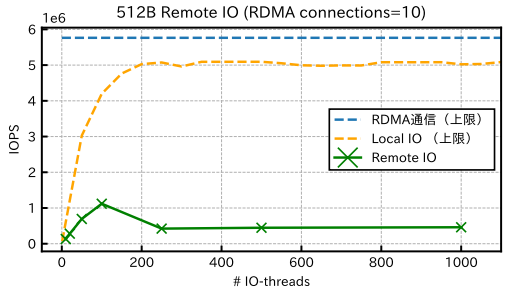

Fig. 10: 512B リモート IO（RDMA コネクションを 10 に固定）

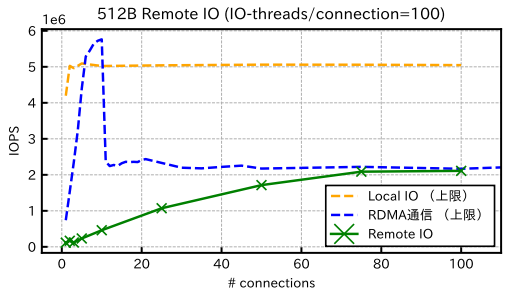

Fig. 11: 512B リモート IO（コネクション当たりの IO スレッド数を 100 に固定）

次に，2 つのパラメータのどちらかを固定することなく，様々なパラメータで実験を行った．結果はリードサイズ別にそれぞれ，Fig. 12，Fig. 13である．4KiB の結果（Fig. 12）では，IO スレッド数が 1000 付近でピーク性能に達している一方で，同じ IO スレッド数でも性能にバラツキがある．このことは，コネクション数によって性能が大きく変わるということを示しており，リモート IO を用いてパフォーマンスを追求する際は，パラメータに細心の調整が必要なことがわかる．64KiB の結果（Fig. 13）では、系全体の IO スレッド数によってほぼ一意に性能が定まっている．IO スレッド数をある程度確保しておけば，ほぼ最大性能値を達成できるということであり，リードサイズが 64KiB のときは系としてよりロバストであるといえる．

1：系全体の IO スレッド数とは RDMA のコネクション数とコネクション当たりの IO スレッド数をかけ合わせた値であり，リモートサーバ上に存在する全ての IO スレッドの数である．

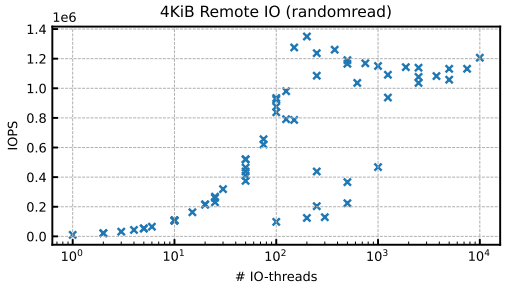

Fig. 12: 4KiB リモート IO

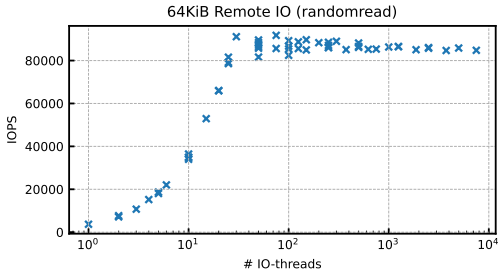

Fig. 13: 64KiB リモート IO

4.3.4 RDMA 通信，ローカル IO，リモート IO のまとめ

リモート IO と，リモート IO を構成するローカル IO と RDMA 通信の性能測定を行い考察を行った．最後に，リードサイズ別の各方式の最大性能値を纏めたものが，Fig. 14である [2]．ここでは IOPS ではなくスループットによる比較であることに注意されたい．リードサイズが 512B のときは，4KiB 以上のときと比較して性能が劣っている．これは，実験環境の OS のページサイズ設定が 4KiB であることにより，512B リードは内部的には 4KiB リードに変換されていたためであると考えられる．

本実験の主題であるリモート IO について着目すると，4KiB 以上のリードサイズでは，RDMA 通信がボトルネックとなっていることがわかり，リモート IO でもほぼ同様の性能を達成している．原理上これよりスループットの高いアクセスは，RDMA 通信の方式から抜本的に変更しないと原理上不可能であり，理想的な結果が得られたといえる．

Disaggregated storage を考えたときに，ローカル IO とリモート IO のアクセスコストの差が重要になる．そこでリモート IO がローカル IO と比較してどれだけの IOPS を達成したかを，Table 2に纏めた．512B では理論的性能最大値（この場合ローカル IO の 5.1M IOPS）の 85%である 4.3M IOPS を達成している．これはリモートストレージへのアクセスであることを考えると非常に高速である．

4.4 IO マイクロベンチマーク（IO リプレイ）

4.4.1 実験設定

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．

IO リプレイでは IO を再現するだけで，select，where，groupby，orderby などの演算は行わない．

まず IO スレッド数を 10000 に固定してローカル IO とリモート IO 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 Fig. 15である．Table 2によればリードサイズが 16KiB のときランダムリードにおいて，リモート IO の IOPS はローカル IO の IOPS の 55%であるので，実行時間は高々約 2 倍になるはずだが，実際はもっと長くなっている．これは，リモート IO では RDMA にキャッシュ機構がないために，ローカル IO と比較してキャッシュが効かず，IO アクセスパターンに局所性がある IO リプレイでは，性能差がランダムリードのときより開いているからであると考えられる．

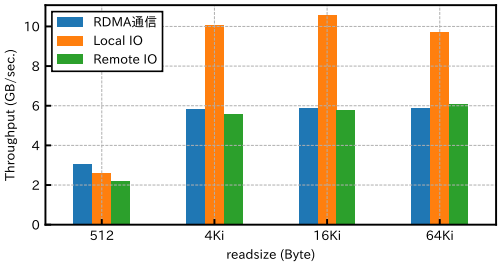

Fig. 14: リードサイズ別の各方式のランダムリード性能最大値

Table 2: リードサイズ別のローカル IO，リモート IO の IOPS 比較

リードサイズ	ローカル IO	リモート IO	$\frac{\text{リモート } IO}{\text{ローカル } IO}$
512 B	5.1 M	4.3 M	85%
4 KiB	2.5 M	1.4 M	55%
16 KiB	0.64 M	0.35 M	55%
64 KiB	0.15 M	9.3 M	62%

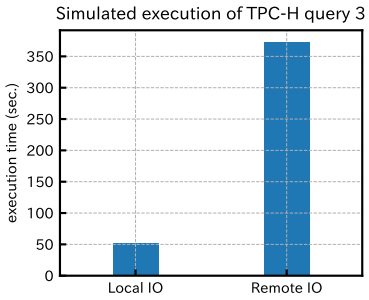

Fig. 15: 16KiB での IO リプレイのローカル IO とリモート IO の実行時間の比較

2：ここでは紙面の都合上，前節では掲載できなかった 512B,16KiB,64KiB リードサイズの結果も掲載している．

そこでより詳しく，IO リプレイでの性能特性を見るために，ローカル IO，リモート IO のそれぞれで IO スレッド数を変化させて挙動を確かめた．

4.4.2 IO リプレイ（ローカル IO）の性能測定

ローカル IO の結果はリードサイズ別にそれぞれ Fig. 16, Fig. 17である．参考としてランダムリードの結果も併せて載せる．やはりローカル IO では IO リプレイのほうが大幅に性能が良い．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．

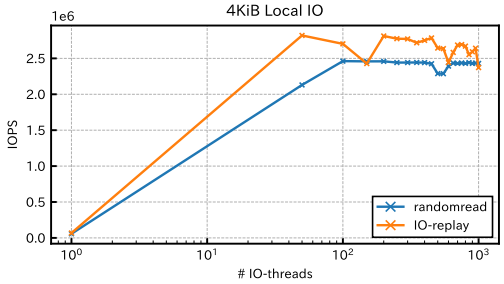

Fig. 16: ランダムリードと IO リプレイに於ける 4KiB ローカル IO

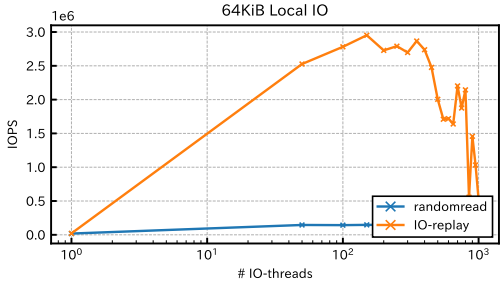

Fig. 17: ランダムリードと IO リプレイに於ける 64KiB ローカル IO

4.4.3 IO リプレイ（リモート IO）の性能測定

コネクションあたりの IO スレッド数を固定して，コネクション数を変化させてプロットすると，リードサイズ別に Fig. 18, Fig. 19のようになる．リモート IO では，IO リプレイのほうがランダムリードとして比較して，性能が優位に良いという現象は見られない．リモート IO ではキャッシュが効かないということを考えると，妥当な結果である．

4.5 IO リプレイによるクエリ実行

4.5.1 実験設定

実験用のデータセットとして TPC-H を使用した．データ生成には TPC-H に付属するツールである dbgen によって生成した．dbgen では Scale Factor (SF) と呼ばれる値を設定することで，生成するデータ量を調整することが出来，本研究では SF=1 に設定した．

TPC-H では本来複数のテーブルが定義されているが，今回

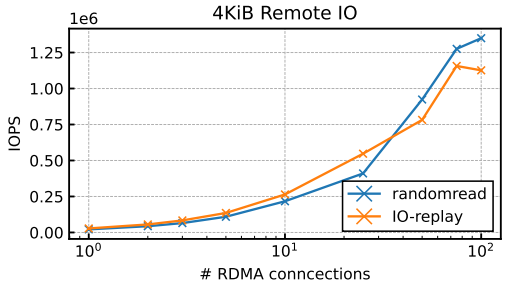

Fig. 18: ランダムリードと IO リプレイに於ける 4KiB リモート IO

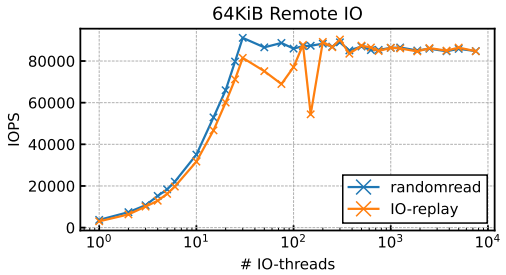

Fig. 19: ランダムリードと IO リプレイに於ける 64KiB リモート IO

は用途に合わせて lineitem テーブルを使用した．なお SF=1 で生成された lineitem は Table 3のとおりである．

Table 3: dbgen によって生成された lineitem

Scale Factor	1
レコード数	6001215
サイズ	759863287B

また，TPC-H ではベンチマークのためのクエリも複数用意されている．lineitem テーブルのみを使用するクエリである，クエリ 1，クエリ 6，クエリ 15 を使用した [3]．クエリ 15 に関しては lineitem テーブルのみを使用するのはサブクエリであったのでサブクエリのみの実行となっている．なお，select，groupby，orderby のような演算は今回は行っておらず，where 句のみの演算である．

なお，予めクエリを実行し，各クエリの選択率を求めたところ Table 4のようになった．選択率が取りうる値は [0,1] であるので，多種の選択率が存在しており実験設定として妥当であることがわかる．

Table 4: TPC-H の各クエリの選択率

クエリ	選択率
1	0.97
6	0.013
15	0.15

3：これらのクエリは選択率が大きく異なるが，結果は凡そ同一であったので，結果の掲載はクエリ 1 のみとする

4.5.2 クエリ 1 の性能測定

選択率が 0.97 であるクエリ 1 を実行した結果を示す．

まず，系全体の IO スレッド数と性能の関係を Fig. 20に示す．縦軸は実行に要した時間であり，短いほど性能が良いといえる．ローカルクエリとリモートクエリ（リモート処理）では IO スレッドの数によらず，比較的安定して高性能を達成していることがわかる．一方，リモートクエリ（ローカル処理）では IO スレッド数が少ない領域でこそ安定して高性能だが，IO スレッド数が増加すると性能が悪いケースが多くなっていることがわかる．

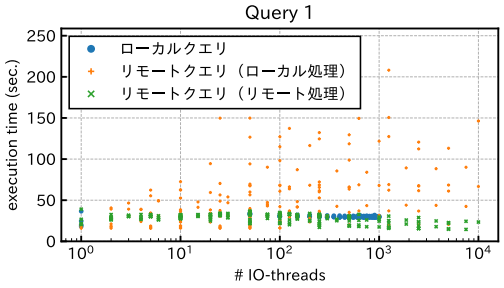

Fig. 20: クエリ 1 の IO スレッド数と実行時間

次に，RDMA コネクション数と性能の関係を Fig. 21に示す．ローカルクエリでは RDMA を使用しないので，このグラフには示されない．リモートクエリ（リモート処理）では，IO スレッド数のときと同様に，RDMA コネクション数によらず，安定して高性能を達成している．対して，リモートクエリ（ローカル処理）では RDMA コネクション数が増加するほど性能が悪化している．

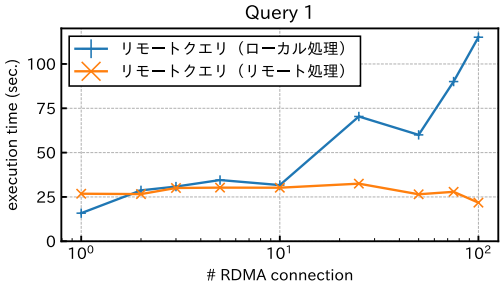

Fig. 21: クエリ 1 の RDMA コネクション数と実行時間

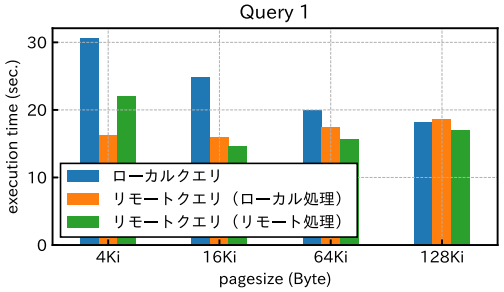

Fig. 22: クエリ 1 のページサイズと最短実行時間

次に，ページサイズ別で，最良の実行時間であったケースを方式ごとに Fig. 22で比較した．（それぞれの場合でベストなケースを抽出しているので，それぞれ IO スレッド数と RDMA コネクション数は異なる．）ページサイズを大きめに取るほど各方式ごとの性能差が小さくなることがわかる．ページサイズが 128KiB の場合は，性能は各方式でほぼ同一で，クエリをどのサーバで処理するかはほとんど気にしなくてよいと言える．ページサイズ 4KiB のとき以外は，リモートクエリ（リモート処理）の性能が最良である．直感的には RDMA によるデータ転送を必要としないローカルクエリの性能が最も良さそうであるが，実際は真逆である．

この理由を探るため，ワークロード実行中の CPU 使用率を監視したところ，性能が良いリモートクエリ（リモート処理）ではワークロード実行中の CPU 使用率が両サーバで 100%になっており，しっかり両サーバの全コアが使い切られていることがわかった．一方，ローカルクエリ，リモートクエリ（ローカル処理）では CPU 使用率が低く，あまりコアが使われていないことがわかった．

これらの結果を総合して考えると，ローカルクエリ，リモートクエリ（ローカル処理）ではマルチスレッドで多重化せずに，シングルスレッドで IO スレッド数，RDMA コネクション数ともに 1 にしておくのが性能上良いといえる．リモートクエリ（リモート処理）ではベストな性能を求める場合，多重化したほうがよいが，多重化しなくてもそれほど性能に大差はないといえる．

5 関 連 研 究

第 1節で述べたように，RDMA を用いたリモート IO 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，どのようにデータベースシステムに応用できるか数多くの研究がなされている．

RDMA を採用した分散データベースシステム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 [2–6]．例えば [25] 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サーバ間通信でありメモリ上で完結する故に，ストレージまで IO 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 IO のコストが下がった結果，ローカル IO とのアクセスコストの差が減少し，Disaggregated storage （非集約型ストレージ）上での IO のアクセスコストのバランスが従来と比べて変化した．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．

6 おわりに

Disaggregated storage architecture（非集約型ストレージアーキテクチャ）上に並列データベースを構成する際，データがどのストレージに格納されているかによってアクセスコストが異なることがパフォーマンスを追求する上で課題であった．本研究では，まず潜在的最大性能を引き出す IO の発行方法を実験により明らかにした．特にリモート IO の場合では，RDMA と IO をリモートサーバで高速にブリッジングする必要があったがこれを達成するシステムソフトウェアを考案した．これにより，ローカルのストレージだけでなく，リモートストレージにも高速でアクセス出来るようになり，最大でローカル IO の場合 5.1M IOPS，リモート IO の場合 4.3M IOPS を達成し，系全体で IO のコストが低下した．

次に，これらの明らかになった IO の性能を考慮した上で，オペレータをどのようにサーバに配置してクエリ処理をするか 3 つの方式を提案し，実装を用いた実験を行った．リモートクエリ（リモート処理）ではローカルサーバ，リモートサーバの CPU を使い切る事ができる一方で，ローカルクエリやリモートクエリ（ローカル処理）では CPU を使い切る事ができず，実験の結果直感に反して，多くの条件下でリモートクエリ（リモート処理）の性能がローカルクエリの性能を上回ることが明らかになった．

今後の課題として，クエリ実行の実験において，同時に複数クエリを処理する状況における性能検証が挙げられる．

謝　辞

本研究の一部は、日本学術振興会科学研究費補助金 20H04191 の助成を受けたものである。

文　献

[1] F. J. Hens and J. M. Caballero, *Triple Play: Building the converged network for IP, VoIP and IPTV*, vol. 3. John Wiley & Sons, 2008.

[2] 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.

[3] 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.

[4] 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.

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

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

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

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

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

[19] S. Babu and H. Herodotou, “Massively parallel databases and mapreduce systems,” *Foundations and Trends® in Databases*, 2013.

[20] M. Stonebraker, “The case for shared nothing,” *IEEE Database Eng. Bull.*, vol. 9, no. 1, pp. 4–9, 1986.

[21] E. Rahm, “Parallel query processing in shared disk database systems,” *ACM SIGMOD Record*, vol. 22, no. 4, pp. 32–37, 1993.

[22] G. K. Lockwood, D. Hazen, Q. Koziol, R. S. Canon, K. Antypas, J. Balewski, N. Balthaser, W. Bhimji, J. Botts, J. Broughton, *et al.*, “Storage 2020: A vision for the future of hpc storage,” 2017.

[23] S. Legtchenko, H. Williams, K. Razavi, A. Donnelly, R. Black, A. Douglas, N. Cheriere, D. Fryer, K. Mast, A. D. Brown, A. Klimovic, A. Slowey, and A. Rowstron, “Understanding Rack-Scale disaggregated storage,” in *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17)*, (Santa Clara, CA), USENIX Association, July 2017.

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

[25] 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.