分散KVSにおけるメニーコアを活用したプロキシによるクエリ集約及び排他制御

三輪 竜也[1]　川浪 大知[1,†1]　川島 龍太[1]　松尾 啓志[1]

概要：ビッグデータの普及により，大規模データの保存や高速な検索処理が可能な分散キーバリューストアが一般的となった．Cassandra は，全ての KVS ノードがクライアントからの要求を受け付けることで，高可用性やスケーラビリティを担保しつつ，高スループットを実現できる．しかし，P2P 型の分散キーバリューストアは，その構成上トランザクションや排他制御を高速に行うことが困難である．我々は，中規模分散キーバリューストアを想定し，メニーコアを活用した集中型のプロキシによるクエリ集約手法を提案している．しかし，従来のプロキシでは，クエリ処理のプロセスと TCP/IP プロトコルスタックのプロセス間でパケットデータのコピーが発生する実装となっていた．また，NIC のマルチキュー機能を想定しないスレッド構成になっていたため，ネットワーク処理がボトルネックになっていた．さらに，クエリのルーティングを行うクライアントスレッドとクエリの集約・分解を行うスレッド間のデータの受け渡しに用いるキューに mutex による排他制御を使用していたため，大きなオーバヘッドが発生していた．そこで本研究では，プロトコルスタックの処理スレッドを複数動作させることでメニーコアを活用し，ネットワーク処理を改善しつつ，そのスレッド上にクエリルーティング処理やクエリ集約・分解処理を実装することで，プロセス間通信やパケットコピーの削減を行う．また，クライアントスレッドと集約・分解スレッド間のデータ受け渡しにロックフリーなキューを利用することでプロキシのさらなる高速化を実現する．さらに，プロキシがクライアントからの要求を全て受け付け，ロックの管理を行うことで，排他制御に必要な KVS ノード間の通信を削減する．性能評価の結果，クライアント及び KVS ノードに DPDK を用いる条件下において，プロキシは 10 Gbps のスループットを達成し，さらにプロキシによる集中的な排他制御によって，Paxos を用いた場合と比較して，レイテンシが約 35 分の 1 となった．

キーワード：分散キーバリューストア，クエリ集約，排他制御，DPDK，Paxos

1. はじめに

代表的な分散キーバリューストア (Distributed Key Value Store: D-KVS) である Apache Cassandra [1] では，全ての KVS ノードがクライアントからの要求を受け付けることで，高可用性やスケーラビリティを担保しつつ，高スループットを実現できる．しかし，D-KVS では KVS ノード間でのクエリの転送により，パケットのネットワーク処理回数が増加する。ネットワーク処理にはコンテキストスイッチなどの処理が発生するため，D-KVS でのクエリ転送によるネットワーク処理回数の増加は大きなオーバヘッドとなる．また，全ての KVS ノードがクライアントからの要求を受け付けるため，KVS ノード間でのデータの一貫性保証が必要である．これは，Paxos [2] などの分散合意プロトコルを用いることで実現可能であるが，データ一貫性保証のためのノード間通信が多発し，排他制御やトランザクションを高速に行うことが困難である [3].

先行研究 [4] では，中規模 D-KVS を想定し，メニーコアを活用したプロキシによるクエリ集約手法を提案した．プロキシでクエリ集約を行うことで，D-KVS でのクエリ転送によるネットワーク処理回数を削減するとともに，プロキシがクライアントからの要求を全て受け付けることで，D-KVS における排他制御のための通信を削減可能となる．しかし，従来のプロキシでは，クエリ処理のプロセスと TCP/IP プロトコルスタックのプロセス間でパケットデータのコピーが発生する実装となっていた．また，NIC のマルチキュー機能を想定しないスレッド構成になっていたため，ネットワーク処理がボトルネックとなりスループット向上が困難であった．さらに，クエリのルーティングを行うクライアントスレッドとクエリの集約・分解を行

[1] 名古屋工業大学大学院
Nagoya Institute of Technology
[†1] 現在，株式会社 ボスコ・テクノロジーズ
Presently with BOSCO Technologies Inc.

© 2019 Information Processing Society of Japan

うスレッド間のデータの受け渡しに用いるキューに mutex による排他制御を使用していたため大きなオーバヘッドが発生していた．また，プロキシを用いた排他制御は未実装であった．

そこで本研究では，クエリ集約によるプロキシのさらなる高速化および排他制御の実装を行う．プロトコルスタックの処理スレッドを複数動作させることでメニーコアを活用し，ネットワーク処理を改善しつつ，そのスレッド上にクエリルーティング処理やクエリ集約・分解処理を実装することで，プロセス間通信やパケットコピーの削減を行う．また，クライアントスレッドと集約・分解スレッド間のデータの受け渡しに CAS を用いたロックフリーなキューを利用することでプロキシのさらなる高速化を実現する．さらに，プロキシ一台でロックの管理を行うことで，KVS ノード間における一貫性保証のための通信を削減し排他制御を効率化する．

本稿の構成は以下の通りである．まず第 2 章では D-KVS の通信処理における問題点や排他制御における問題点について説明する．次に第 3 章で関連研究について述べ，第 4 章で先行研究におけるクエリ集約およびプロキシの実装における問題点について説明する．第 5 章で提案手法とその実装について述べ，第 6 章で提案手法の性能比較と考察を行い，第 7 章でまとめと今後の課題を述べる．

2. D-KVS における問題点

Amazon Dynamo [5]，Apache Cassandra，Riak [6] など多くの D-KVS は性能及び可用性向上のために，リングアーキテクチャを用いている．リングアーキテクチャでは，コンシステントハッシング [7] によってデータを分散する．Key の担当ノードはハッシュ値によって決まるため，クエリを実行する前に，クエリ処理の対象となる Key を担当するノードまでクエリをルーティングする必要がある．そのためリングアーキテクチャには，構造上 2 つの問題点が存在する．

2.1 通信処理における問題点

D-KVS では，大量の計算機を並べてスケーラビリティを確保するため，各 KVS ノードは，汎用サーバ上で動作する．汎用サーバ上のネットワークスタックでは，NIC にパケットが到着した場合，割り込み処理によってカーネルによるパケットの処理が行われる．この時，割り込みに伴うコンテキストスイッチがオーバヘッドとなり，ネットワーク帯域を活用できない [8]．また，カーネル空間からユーザ空間へデータをコピーするため，メモリコピーやシステムコール呼び出しに伴う遅延が発生する．さらに，データの冗長化のためにレプリケーションを行うため，パケット数はレプリカ数分だけ増加する．以上のことから，D-KVS の性能を改善するためには，クエリの転送で発生するパケット数を削減し，ネットワークの処理負荷を軽減することが必要となる．

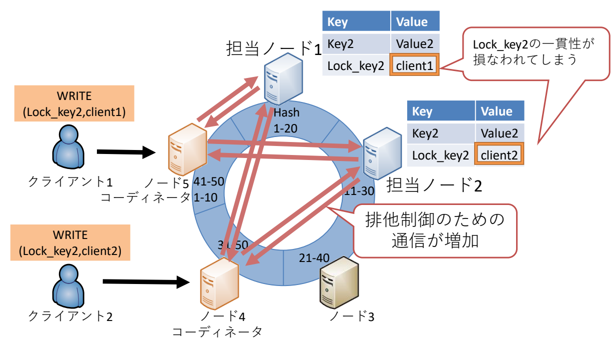

図 1 Paxos による排他制御

2.2 排他制御における問題点

P2P 型の D-KVS は，全ての KVS ノードがクライアントからの要求を受け付けるため，基本的に結果一貫性である．しかし，厳密な一貫性を必要とする場合はデータの強い一貫性保証が必要である．Cassandra や Redis，DynamoDB といった D-KVS では複数のデータをアトミックに更新するため，分散合意プロトコルを用いてトランザクションや排他制御の仕組みを実装している [9] [10] [11]．しかし，代表的な分散合意プロトコルである Paxos では，ノード間で最低 2 往復のメッセージ通信を必要とするため，D-KVS において効率的な排他制御が困難である (図 1)．

3. 関連研究

3.1 パケット集約

NetAgg [12] は MapReduce や検索エンジンにおいてパケット処理量を削減する手法である．複数の計算機から単一の計算機へデータを集約する際に，ネットワーク上で事前に集約を行うことで大量のデータが一台に集中することを回避している．PA-Flow [13] は，NFV 環境で用いられるパケット集約手法である．ファイアウォールなどのネットワークファンクション (NF) を繋げたサービスチェーン上では，上流部分にパケットが集中して性能低下を引き起こす．そのため，各 NF で同一ホップ先のパケットを段階的に集約することで，上流部分でのパケット数を削減し，スループットを向上させている．

3.2 排他制御

排他制御を高速に行うために，分散合意プロトコルを高速化する方法がある．FastPaxos [14], Speculative Paxos [15], NoPaxos [16] では，ネットワーク上でメッセージの到着順を保証することで，Paxos に必要な通信回数の一部を削減している．しかし，これらの手法を利用するには，特殊なネットワークやハードウェアを必要とする．

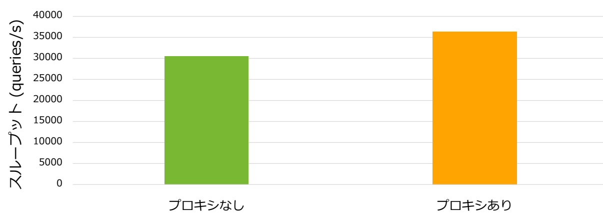

図 2　プロキシによるスループットの向上 [4]

4.　先行研究

先行研究では，中規模 D-KVS を想定し，メニーコアを活用した集中型のプロキシによるクエリ集約手法を提案した [4]．プロキシでクエリ集約を行うことで，D-KVS でのクエリ転送によるネットワーク処理回数を削減する．また，プロキシがクライアントからの要求を全て受け付けることで，D-KVS における排他制御のための通信を削減することができる．しかし，プロキシを用いた排他制御は未実装であった．

4.1　プロキシの性能

先行研究では，クライアントと D-KVS の間にプロキシを導入しクエリパケットの集約を行うことで性能向上を図る．YCSB [17] および KVS ノードに Cassandra を用いた性能評価では，図 2 に示すように，プロキシを用いた場合 Cassandra のスループットが最大 19%向上した．しかし，計測したプロキシのスループットは，10 Gbps のネットワーク速度の 1/200 程度である．これは従来のプロキシの実装に問題点があるためである．したがって，本稿ではプロキシの実装の改善を行う．

4.2　クエリ集約手法

プロキシ上でクエリパケットの集約を行うことで KVS ノードが処理するパケット数を削減する (図 3)．プロキシはクライアントから送信されたクエリを一つのパケットに集約し，それを担当ノードへ送信することで送信パケット数を削減し，D-KVS でのクエリ転送によるネットワーク処理回数を削減する．

4.3　従来のプロキシの実装における問題点

従来のプロキシの実装方式を図 4 に示す．従来のプロキシでは，TCP/IP プロトコルスタックとして DPDK-ANS [18] を使用し，クエリのルーティングや集約を行うアプリケーションプロセスとは別のプロセスとして動作するような実装であった．したがって，ネットワーク処理プロセスとアプリケーションプロセスは ANS の提供するソケット API を使用して通信を行う．そのため，プロセス間通信が必要となり，ソケット API を用いることでプロトコルスタックからクエリの処理を行うプロセス間でパケットデータのコピーが発生する．また，NIC のマルチキュー機能を想定しないスレッド構成になっていたため，ネットワーク処理がボトルネックとなっていた．

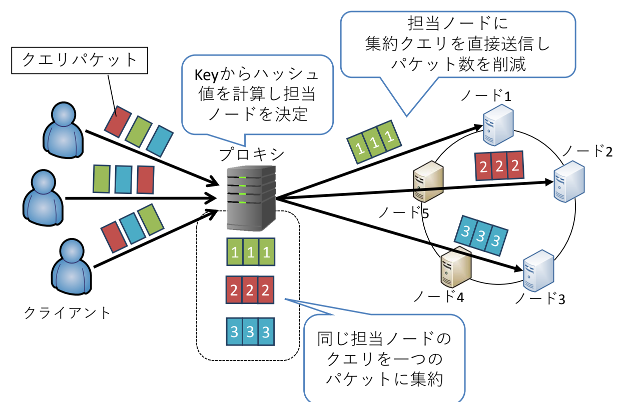

図 3　プロキシでのクエリ集約手法

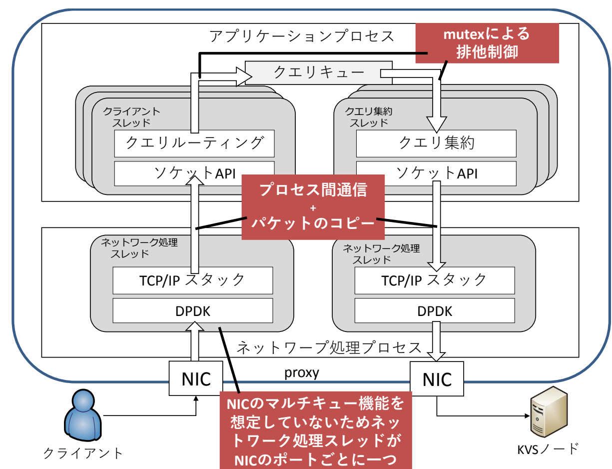

図 4　従来のプロキシの実装

クライアントスレッドは，クライアントとの通信およびクライアントから受け取ったクエリを集約・分解スレッドに受け渡す役割を持つ．クライアントスレッドは，接続されるクライアント毎に生成され，処理が完了したクエリの通知を受け取るためのリザルトキューを持つ．集約・分解スレッドは，同一宛先となるクエリを集約し，KVS ノードに転送する役割を持つ．集約・分解スレッドは宛先となる KVS ノード毎に生成され，各スレッドがクエリを受け取るためのクエリキューを持つ．クライアントスレッドや集約・分解スレッドはそれぞれ複数生成することが可能な実装となっており，スレッド間で処理負荷を分散させている．クライアントスレッドは接続されるクライアント毎に生成されるため，クライアント数がプロキシの CPU コア数を超えた場合は，カーネルによってスケジューリングされるため，スレッドのコンテキストスイッチが発生する．また，スレッド間のデータの受け渡しに用いるキューには，mutex による排他制御を使用していたため，クエリ毎に mutex の獲得や解放処理が必要となる．

© 2019 Information Processing Society of Japan

5. 提案手法

本章では，プロキシにおける TCP/IP プロトコルスタックの処理スレッドを複数動作させることでメニーコアを活用し，ネットワーク処理を改善しつつ，そのスレッド上にクエリのルーティング処理や集約・分解処理を実装することで，プロセス間通信やパケットコピーの削減を行い，さらにクライアントスレッドと集約・分解スレッド間のデータの受け渡しにロックフリーなキューを利用することによるプロキシの高速化を行う．また，プロキシでロック管理を行い，KVS ノード間におけるデータ一貫性保証のための通信削減による排他制御の効率化を提案し，それらの実装について述べる．

5.1 プロキシの高速化実装

先行研究におけるプロキシでは，NIC のマルチキュー機能を想定しないスレッド構成になっていたため，ネットワーク処理がボトルネックとなっていた．さらにクエリ処理プロセスとネットワーク処理プロセスが別で動作していたため，プロセス間通信やパケットコピーが発生する実装となっていた．さらに，クライアントスレッドと集約・分解スレッド間のデータの受け渡しに用いるキューに mutex による排他制御を使用していたため，大きなオーバヘッドが発生する．したがって，それらの改善が必要である．

本研究では，NIC のマルチキュー機能活用によるネットワーク処理の改善およびクエリ処理プロセスとプロトコルスタック処理プロセス統合によるプロセス間通信とパケットコピーの削減を行った．プロキシの高速化実装にあたり TCP/IP プロトコルスタックに高速なパケット処理を実現する DPDK [19] を利用する mTCP [8] を使用した．mTCP はネットワーク処理スレッドを複数動作させることが可能であり，RSS [20] を用いてネットワーク処理を並列化し，従来のプロキシにおけるネットワーク処理の改善が可能である．しかし，mTCP を用いる場合，ネットワーク処理スレッドとクエリ処理スレッドが別で動作するため，スレッド間でソケット API と共有バッファを用いたデータの受け渡しを行う必要があり，データコピーが 2 回発生する．したがって，データコピー削減のため，mTCP のスレッド構成の変更を行った．ネットワーク処理スレッド上にクエリのルーティング処理や集約・分解処理を実装することでスレッド間におけるデータコピーを削減した．さらに，クライアントスレッドと集約・分解スレッドで共有されるクエリキューやリザルトキューにロックフリーなリングキュー (rte_ring) を用いる．rte_ring は CAS 命令を使用して実装されており，mutex を利用したキューよりもキューの操作を高速に実行することができる．

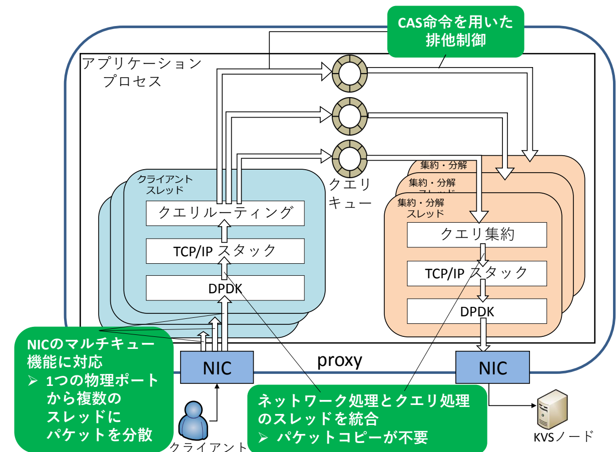

図 5 プロキシの高速化実装

5.2 プロキシのコンポーネント

プロキシにおけるクエリの処理フローを図 6 に示す．

- クライアントスレッド
  NIC の受信キューからクエリパケットを取り出しクエリの宛先を計算する．クエリキューが宛先 KVS ノード毎に用意されているため，クエリの宛先を基にクエリキューに追加する．また，リザルトキューから処理結果を取り出し NIC の送信キューに追加する．クエリを送信するクライアントスレッドとクエリの処理結果を送信するクライアントスレッドは同一スレッドである．これは mTCP の実装上スレッドの分離が困難であったためである．また，mTCP の仕様上，クライアントスレッドは一つの論理コアに一つまでしか動作できない．
- 集約・分解スレッド
  クエリキューからクエリを複数取り出し，それらを一つのパケットに集約し，NIC の送信キューに追加する．また，クエリの処理結果を NIC の受信キューから取り出し，個別の結果に分解する．クエリの集約と分解を行うスレッドは同一スレッドである．これは mTCP の実装上スレッドの分離が困難であったためである．また，mTCP の仕様上，集約・分解スレッドは一つの論理コアに一つまでしか動作できない．
- クエリキューおよびリザルトキュー
  CAS 命令を使用するロックフリーなリングキューであり，クライアントスレッドと集約・分解スレッド間でのデータの受け渡しに用いる．

5.3 プロキシでのロック管理

先行研究ではプロキシを用いた排他制御は未実装であったため，プロキシでのロックの実装を行う．プロキシ一台でクライアントからの要求を受け付け，ロック管理を行うことで，KVS ノード間における排他制御のための通信を削減し，Paxos と同様に強い一貫性を保証する．また，排

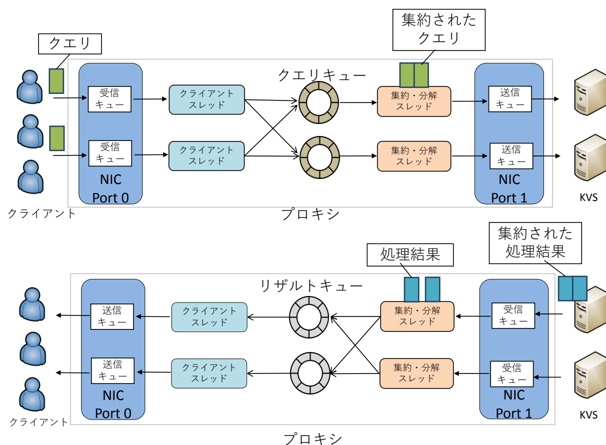

図 6　クエリ処理フロー

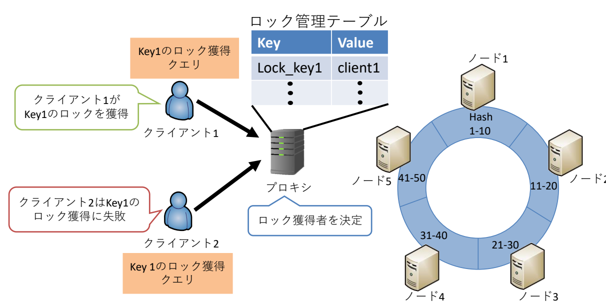

図 7　プロキシでのロック管理

他制御のための通信が削減されるため，Paxos よりも効率的にトランザクションを実現できる．しかし，本論文ではロックの獲得のみに焦点を当てており，トランザクション処理機能の実現には至ってない．ここで，プロキシでのロック管理の様子を図 7 に示す．クライアントが Key のロックの獲得を行う場合，プロキシはロック獲得テーブルにロック獲得者を記憶することで，ロック獲得者を決定する．ロック獲得処理は集約・分解スレッドが行う．ロック管理テーブルは各集約・分解スレッドで共有しているため，レプリカにも対応している．しかし，ロック管理テーブルへのアクセスにおける競合が発生する．そこで，CAS を利用する Test-and-Set 命令を使用することで，スレッドを待ち状態に移行することなくクリティカルセクションを作成し，ロック管理テーブルの排他制御を行う．ロック獲得処理を集約・分解スレッドで行う理由は，ロック管理テーブルへのアクセス競合がクライアントスレッドでロック獲得処理を行う場合よりも少ないためである．

6. 評価実験

提案手法の有効性を確認するために，高速化実装を行ったプロキシおよびクライアントがクエリルーティングを行う場合で性能比較を行った．また，プロキシで排他制御を行う場合と Cassandra でオリジナルの Paxos を用いて排他制御を行う場合で性能比較を行った．

6.1 評価環境

6.1.1 クライアント

クエリ集約の有効性を評価するためのクライアントとして，書き込みクエリを生成し送信を行うプログラムを作成した．多数のクライアントが接続する環境を想定し，クライアントマシンで，クエリの生成・送受信を行うスレッドを 3 つ動作させ，その各スレッドはプロキシに対して 32 個の TCP コネクションを生成する．同様に，クライアントがクエリルーティングを行う評価では，クライアントの各スレッドが KVS ノード毎に 32 個の TCP コネクションを生成する．プロキシでクエリ集約を行うことでクエリパケット数を削減し，ネットワーク処理回数を削減することの有効性を示すため，クエリサイズが小規模となるように，送信するクエリのキーとバリューのサイズはそれぞれ 5 バイトと設定した．クエリサイズは合計で 84 バイトであり，Ethernet フレームのサイズは 142 バイトとなる．評価では，単位時間あたりに正常に受信したクエリ処理の数およびクエリを送信してから結果を受信するまでの時間を計測した．また，プロキシでのロック獲得の有効性を評価するためのクライアントとしてキーとバリューサイズがそれぞれ 5 バイトであるクエリを 100 万個送信するプログラムを作成し，クエリを送信してから結果を受信するまでのロック獲得レイテンシの計測を行った．クライアントは 1 個の TCP コネクションを生成する．

6.1.2 KVS ノード

プロキシの性能評価に使用する KVS として，クエリの受信と評価結果の送信を行うプログラムを作成した．プロキシの性能を評価することが目的であるため，通信処理のみを実装しており，KVS ノードでのデータの永続化は行わない．さらに，パケット処理を高速に行うため DPDK を使用し実装した．また，プロキシでのロック獲得の評価では KVS として Cassandra を用いた．

6.2 評価に用いる計算機

プロキシには表 1 に示す計算機を使用した．また，クライアントと KVS(自作 KVS と Cassandra) には，表 2 に示す計算機を使用した．プロキシの性能評価ではクライアントに 3 台，KVS ノードに 5 台の計算機を使用した．プロキシでのロック獲得の評価では，Cassandra ノードを 5 台使用し，Cassandra で排他制御を行う場合の評価では Cassandra ノード数を 3 台から 9 台まで増加させ評価を行った．この時，レプリケーション数はノード数と同様である．

6.3 プロキシの性能評価

実装したプロキシの性能の測定を行うために，クライアント，プロキシ，KVS ノードを 10 Gbps のネットワークに接続し，スループットおよびレイテンシを計測した．ま

© 2019 Information Processing Society of Japan

表 1　プロキシに用いる計算機

OS	Ubuntu 16.04
CPU	Intel Core i9-7980XE, 2.6 GHz (18 cores / 36 threads)
Memory	64 GB
NIC	Intel X550T (10 GbE, dual port)

表 2　クライアント，KVS ノードに用いる計算機

OS	Ubuntu 16.04
CPU	Intel Core i5-4460 / 3.2 GHz (4 cores / 4 threads)
Memory	16 GB
NIC	Intel X540-T2 (10 GbE, dual port)

た，スループットの評価では，クエリ集約による性能向上を確認するために，プロキシによりクエリ集約を行う方法と，クライアントが担当ノードに直接クエリを送信する方法の 2 つで比較を行った．クエリの集約数はクエリの送信レートによって変動するため，レイテンシはクエリの送信レートの高い場合と低い場合の 2 通りでの評価を行う．評価に用いる計算機は 18 コア/36 スレッドであるため，26 スレッドをクライアントスレッド，10 スレッドを集約・分解スレッドに割り当てた．

まず，クエリの送信レートの高い場合の比較結果を図 8 に示す．グラフの縦軸はスループット，横軸は経過時間を表しており，太線が高速化実装を行ったプロキシを使用した場合，点線がプロキシを使用しない場合を表している．長鎖線は従来の環境で評価した先行研究におけるプロキシのスループットを参考として表示している．破線は 10 Gbps の理論値として 1 クエリが 1 パケットで送信された場合に 1 秒間に送信可能なクエリ数を示している．この評価で用いたプログラムでは，主にクライアントから 1 クエリが 1 パケットで送信されるが，TCP の再送により複数のクエリが 1 パケットで送信される可能性があるため，破線よりもスループットが高くなる場合がある．評価の結果，提案手法が最大 7.45 Mqps となり，10 Gbps 理論値相当のスループットを達成できた．クライアントルーティングでは KVS ノードの受信キューでドロップが観測されたが，提案手法では KVS ノードでのドロップは観測されなかった．クライアントルーティングと比較して提案手法ではスループットが約 30%向上した．また，実装したプロキシを利用した場合の平均クエリの集約数の推移を図 9 に示す．プロキシを用いた場合，平均 2.9 個のクエリが集約されて送信されていた．以上の結果からクエリの集約により KVS ノードでの処理負荷を軽減することができた．

また，この時のレイテンシは表 3 に示す結果となった．提案手法のレイテンシは平均値が 258 倍，中央値が 455 倍に増加した．これは，クエリ集約を行うための処理や，プロキシを経由する 2 hop のルーティングによるものである．KVS ノードが実際に動作する際は，プロキシを用いない場合でも 2hop のルーティングとなるため，実環境ではプロキシなしの場合のレイテンシは評価結果よりも増加することに注意が必要ではあるが，特に高負荷時のレイテンシ増加については今後検討が必要であると考える．

次に，クエリの送信レートを 1 kqps に制限した場合のレイテンシの評価結果を表 4 に示す．提案手法の最小値，中央値は直接送信する場合のレイテンシと比較して 2 倍以内に抑えられている．しかし，平均値は約 8 倍となった．これは，クライアントスレッドから集約・分解スレッドへのクエリの受け渡し処理や 2 hop のクエリルーティングによるものである．

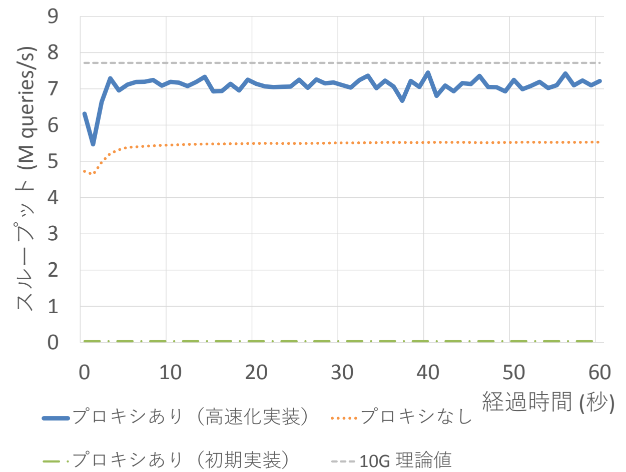

図 8　高負荷時のスループット

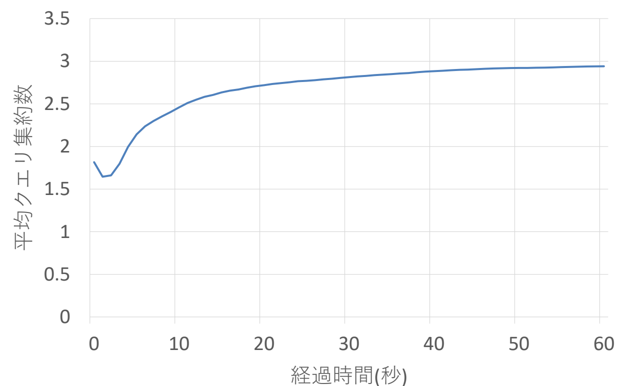

図 9　平均クエリ集約数

表 3　高負荷時レイテンシ

	最小値 (ms)	中央値 (ms)	平均値 (ms)	最大値 (ms)
プロキシなし	0.035	0.48	1.6	2328
プロキシあり (高速化実装)	0.051	217	412	9995

表 4　低負荷時レイテンシ

	最小値 (ms)	中央値 (ms)	平均値 (ms)	最大値 (ms)
プロキシなし	0.043	0.067	0.082	0.63
プロキシあり (高速化実装)	0.057	0.081	0.704	351

© 2019 Information Processing Society of Japan

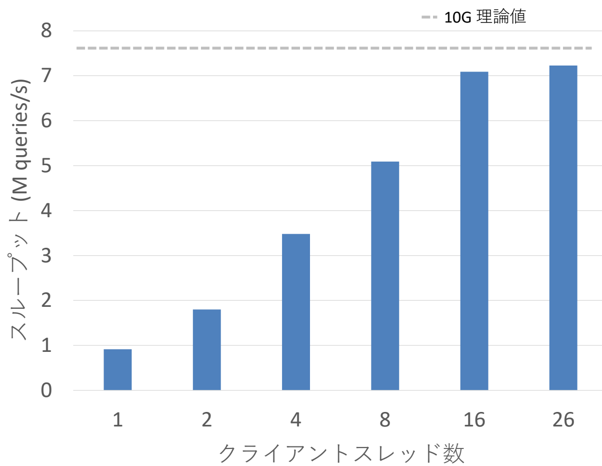

図 10 クライアントスレッド数を変化させた場合の評価

6.4 スケーラビリティ評価

スレッド数を増加させることでプロキシの性能処理が向上することを確認するため，スレッドスを変化させてスループットの測定を行った．実験では，集約・分解スレッド数を 10 に固定し，クライアントスレッド数を 1 から 26 の間で変化させた．評価結果を図 10 に示す．縦軸がスループット，横軸がスレッド数を表す．スレッド数が 1 から 4 までの間では，スループットが線形に増加しているが，スレッド数が 4 から 8 へ変化させた場合は 1.5 倍，スレッド数を 8 から 16 に変化させた場合は 1.4 倍となり，線形には増加していない．また，スレッド数を 26 とした場合では，ネットワーク帯域がボトルネックとなり，性能限界に達している．

スレッド数が 8，16 で性能の向上幅が低下した原因として 2 つ考えられる．1 つ目はスレッド間でデータの受け渡しに用いるキュー操作での競合である．キュー操作には CAS 命令を使用してスレッド間で調停を行っているが，全てのクライアントスレッドが 1 つのキューを同時に操作する可能性があるため，スレッド数が増加するにつれて 1 つのキューに対する競合が増加する．競合を検出したスレッドはキューの操作が成功するまでリトライする必要があるため，スレッド数が増加するほど性能の向上幅が低下する．2 つ目はハイパースレッディングを用いていることである．評価には 18 コア/36 スレッドを用いており，クライアントスレッド数が 8 より大きくなった場合，物理コア以上のスレッド数が動作する．そのため，スレッド数 18 以上ではハイパースレッドを利用して動作するため性能の向上幅が低下したと考える．

6.5 ロック獲得評価

プロキシによるロック獲得レイテンシと Cassandra によるロック獲得レイテンシの比較を行った．評価結果を図 11 に示す．一番左の折れ線グラフはプロキシで排他制御を行った場合，残りの折れ線グラフは Cassandra で Paxos を用いて排他制御を行った場合の結果である．グラフより，ロック獲得クエリの処理をプロキシで行う場合が一番レイテンシが小さい．これは，プロキシ一台でロックの管理を行うことで Cassanda ノード間の一貫性制御のための通信を削減できたためである．また，プロキシで排他制御を行うことで，Paxos を用いた排他制御と比較して，レイテンシが約 35 分の 1 になった．以上より，提案手法であるプロキシを用いた排他制御の有効性を確認できた．

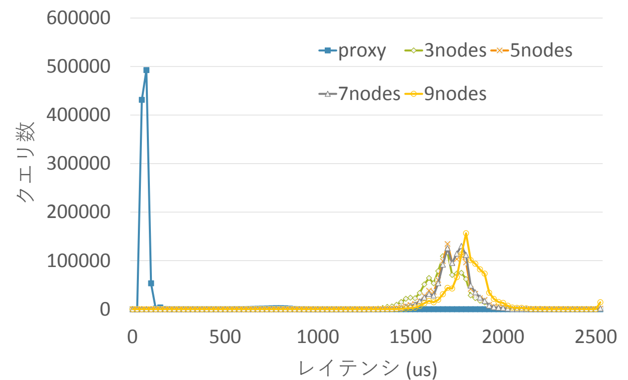

図 11 ロック獲得レイテンシの比較

7. まとめと今後の課題

本論文では，D-KVS に導入されたプロキシの性能改善およびプロキシを用いた排他制御を行った．プロキシの TCP/IP プロトコルスタックに mTCP を使用し，プロトコルスタック処理とクエリ集約処理が同一スレッドで動作するように拡張することで，プロセス間通信やパケットコピーを削減しプロキシの性能向上を行った．また，Key のロック管理をプロキシ一台で行い D-KVS における排他制御のための通信を削減することで排他制御の効率化を行った．

今後の課題として，D-KVS におけるトランザクションの実現やプロキシが単一故障点となる問題の解消などがある．

謝辞 本研究の一部は，科研費基盤研究（C）18K11324 による．

参考文献

[1] Lakshman, A. and Malik, P.: Cassandra: a decentralized structured storage system, *ACM SIGOPS Operating Systems Review*, Vol. 44, No. 2, pp. 35–40. NY, USA, January, 2010.

[2] Lamport, L.: The part-time parliament, *ACM Transactions on Computer Systems (TOCS)*, Vol. 16, No. 2, pp. 133–169. NY, USA, May, 1998.

[3] Verbitski, A., Gupta, A., Saha, D., Brahmadesam, M., Gupta, K., Mittal, R., Krishnamurthy, S., Maurice, S., Kharatishvili, T. and Bao, X.: Amazon aurora: Design considerations for high throughput cloud-native relational databases, *Proceedings of the 2017 ACM International Conference on Management of Data*, ACM,

© 2019 Information Processing Society of Japan

pp. 1041–1052. IL, USA, May, 2017.

[4] Kawanami, D., Kamoshita, M., Kawashima, R. and Matsuo, H.: A Proxy-Based Query Aggregation Method for Distributed Key-Value Stores, *2018 6th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, IEEE, pp. 78–83. Barcelona, ES, August, 2018.

[5] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W.: Dynamo: amazon's highly available key-value store, *ACM SIGOPS operating systems review*, Vol. 41, No. 6, ACM, pp. 205–220. NY, USA, January, 2007.

[6] "Key Value Database — NoSQL Key Value Database — Riak KV — Riak".riak. https://riak.com/products/riak-kv/,(参照 2019-06-01).

[7] Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M. and Lewin, D.: Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web, *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, ACM, pp. 654–663. TX, USA, May, 1997.

[8] Jeong, E., Woo, S., Jamshed, M. A., Jeong, H., Ihm, S., Han, D. and Park, K.: mTCP: a Highly Scalable User-level TCP Stack for Multicore Systems., *NSDI*, Vol. 14, pp. 489–502. Seattle, USA, April, 2014.

[9] "Using lightweight transactions".DATASTAX. https://docs.datastax.com/en/cql/3.3/cql/cql_using/useInsertLWT.html,(参照 2019-06-22).

[10] "トランザクション Redis Documentation (Japanese Translation)".Redis Documentation (Japanese Translation). https://redis-documentasion-japanese.readthedocs.io/ja/latest/topics/transactions.html,(参照 2019-06-01).

[11] "Amazon DynamoDB Transactions".aws. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/transactions.html,(参照 2019-06-01).

[12] Mai, L., Rupprecht, L., Alim, A., Costa, P., Migliavacca, M., Pietzuch, P. and Wolf, A. L.: NetAgg: Using middleboxes for application-specific on-path aggregation in data centres, *Proc. 10th ACM International on Conference on emerging Networking Experiments and Technologies*, ACM, pp. 249–262. Sydney, AUS, December, 2014.

[13] Taguchi, Y., Kawashima, R., Nakayama, H., Hayashi, T. and Matsuo, H.: PA-Flow: Gradual Packet Aggregation at Virtual Network I/O for Efficient Service Chaining, *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, IEEE, pp. 335–340. 2017.

[14] Lamport, L.: Fast Paxos, *Distributed Computing*, Vol. 19, No. 2, pp. 79–103. October, 2006.

[15] Ports, D. R., Li, J., Liu, V., Sharma, N. K. and Krishnamurthy, A.: Designing distributed systems using approximate synchrony in data center networks, *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pp. 43–57. CA, USA, May, 2015.

[16] Li, J., Michael, E., Sharma, N. K., Szekeres, A. and Ports, D. R. K.: Just Say NO to Paxos Overhead: Replacing Consensus with Network Ordering, *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, USENIX Association, pp. 467–483. GA, USA, November, 2016.

[17] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R. and Sears, R.: Benchmarking cloud serving systems with YCSB, *Proceedings of the 1st ACM symposium on Cloud computing*, ACM, pp. 143–154. IN, USA, June, 2010.

[18] "GitHub - ansyun/dpdk-ans: ANS (Accelerated Network Stack) on DPDK, DPDK native TCP/IP stack".GitHub. https://github.com/ansyun/dpdk-ans,(参照 2019-06-22).

[19] "DPDK (Data Plane Development Kit)".DPDK. https://www.dpdk.org,(参照 2019-06-22).

[20] Corporation, M.: Scalable Networking: Eliminating the Receive Processing Bottleneck—Introducing RSS, *Microsoft WinHEC*. April, 2004.