DPDKのRun-to-Completionモデルを用いたL2分散計算環境の提案

山本 竜也[1]　川島 龍太[1]　松尾 啓志[1]

概要：分散計算環境で実行される処理の中には，与えられたデータを繰り返し用いながら，小さいデータを計算機間でやりとりする処理がある．例えば，単回帰分析やロジスティック回帰などの機械学習である．これらの処理において，TCP/IP による制御は相対的に大きなオーバーヘッドとなる．また，カーネルによるパケット I/O 処理は DPDK のパケット I/O 処理に比べて低速である．本研究では，DPDK の Run-to-Completion モデルを用いた L2 分散計算環境を提案する．計算機間の通信に DPDK による L2 通信を用いることによって，TCP/IP による通信のオーバーヘッドを低減するとともに，カーネルによるパケット I/O 処理に比べて高速なパケット I/O 処理を用いることができる．また，Run-to-Completion モデルを採用することによって，Pipeline モデルを用いる場合に比べて CPU リソースや L1 キャッシュを有効活用できる．送られてきた 32 個の値を加算し送り返すプログラムを用いた事前実験 2 では，提案手法を用いた場合の処理性能は TCP/IP による通信を用いた場合に比べて，内部ループ回数が 1 回の場合は 30 倍，10 回の場合は 4 倍，100 回の場合は 0.4 倍高いことを確認した．また，単回帰分析を行うプログラムを用いた評価では，提案手法を用いた 2 台での分散処理の実行時間は，1 台での集中学習の場合に比べて 2 倍，TCP/IP による通信を用いた 2 台での分散処理の場合に比べて 1.2 倍高速であることを確認した．

キーワード：分散計算環境，DPDK，Run-to-Completion，L2 通信，L1 キャッシュ

1. はじめに

分散計算環境で実行される処理の中には，与えられたデータを繰り返し用いながら，小さいデータを計算機間でやりとりする処理形態が存在する．例えば，単回帰分析やロジスティック回帰などの機械学習である．これらの処理において，TCP/IP による制御は計算機内での演算処理時間に比べて，かなり大きなオーバーヘッドとなる．また，カーネルによるパケット I/O 処理は，一定時間に受信するパケットの量が増えると，コンテキストスイッチが増加して，割り込み以外の処理が実行できなくなるため，DPDK のパケット I/O 処理に比べて低速である．

DPDK には Run-to-Completion モデルと Pipeline モデルの 2 つのモデルがある．Run-to-Completion モデルは受信処理，パケット処理，送信処理を一つの論理コアで行い，Pipeline モデルは受信処理，パケット処理，送信処理をそれぞれ別の論理コアで行う．Pipeline モデルはそれぞれの処理が論理コアを専有するため，CPU リソースや L1 キャッシュを有効活用できない．

[1] 名古屋工業大学大学院
Nagoya Institute of Technology

そこで本研究では，DPDK の Run-to-Completion モデルを用いた L2 分散計算環境を提案する．計算機間の通信に DPDK による L2 通信を用いることによって，TCP/IP による通信のオーバーヘッドを低減するとともに，カーネルによるパケット I/O 処理に比べて高速なパケット I/O 処理を用いることができる．また，Run-to-Completion モデルを採用することによって，Pipeline モデルを用いる場合に比べて CPU リソースや L1 キャッシュを有効活用できる．

なお，本研究が提案する分散計算環境には 3 つの前提を設ける．1 つ目の前提は L2 通信が可能であるローカルなクラスタ環境で動作することである．2 つ目の前提は計算機間でやりとりされるデータのサイズは L2 フレームより小さいことである．3 つ目の前提は各計算機で実行される処理はイテレーションが多用され，かつ，その実行に必要なデータは小規模であることである．これらの前提は特に機械学習では満たされることが多いと考える．

本稿の構成は以下のとおりである．第 2 章で分散計算環境の問題点を述べ，第 3 章で DPDK について説明する．第 4 章で提案手法について述べ，第 5 章，第 6 章，第 7 章で提案手法の事前実験と評価を行う．第 8 章で関連研究を

述べ，第 9 章でまとめと今後の課題を述べる．

2. 分散計算環境の問題点

本章では，分散計算環境の問題点として TCP/IP による制御とカーネルによるパケット I/O 処理について述べる．

2.1 TCP/IP による制御

TCP/IP による通信ではルーティング，誤り制御，順序制御といった制御が行われる．ルーティングとは端末の相互接続関係や各端末間の通信回線の混み具合の情報を取得し，送信元端末から宛先端末までのルートを決定する制御である．本研究が提案する分散計算環境は L2 通信が可能であるローカルなクラスタ環境で動作するため，ルーティングは必ずしも必要ではない．誤り制御とはデータが伝送中に誤ったり失われたりしたとき，それを回復して受信側に正しいデータを送り届ける制御である．本研究が提案する分散計算環境の各計算機で実行される処理はイテレーションが多用され，多少のパケットロスであれば処理結果に影響を与えない，また明らかな異常値は処理時に排除することも可能であるため，誤り制御も必ずしも必要ではないと考える．順序制御とは受信データの重複をなくし，正しい順序に並び替える制御である．本研究が提案する分散計算環境の計算機間でやりとりされるデータのサイズは L2 フレーム（ジャンボフレームを含む）より小さいため，順序制御も必ずしも必要ではない．

よって，TCP/IP による制御は本研究が提案する分散計算環境においてはオーバーヘッドであるため，本研究は TCP/IP による通信ではなく L2 通信を用いることにした．

2.2 カーネルによるパケット I/O 処理

Linux2.6 以前のカーネルによるパケット I/O 処理（図 1）では，パケットを受信するたびに NIC（Network Interface Card）からのハードウェア割り込みが発生する．そのため，一定時間に受信するパケットの量が増えると，コンテキストスイッチが増加して，割り込み以外の処理時間が相対的に少なくなる．また，ユーザ空間からカーネル空間にはアクセスできないため，ユーザ空間のアプリケーションがパケットにアクセスするには，受信したパケットをカーネル空間からユーザ空間にコピーしなければならない．

パケット受信時のハードウェア割り込みを削減するために，Linux2.6 以降のカーネルには NAPI（New API）と呼ばれる仕組みが導入された．NAPI ではパケット受信によるハードウェア割り込みが発生すると，NIC からのハードウェア割り込みを一時的に無効化し，NIC のデバイスドライバの挙動を割り込み駆動からポーリング駆動に切り替える．そして，処理すべきパケットがなくなるまでポーリング駆動で処理を行い，通常の割り込み駆動に戻る．

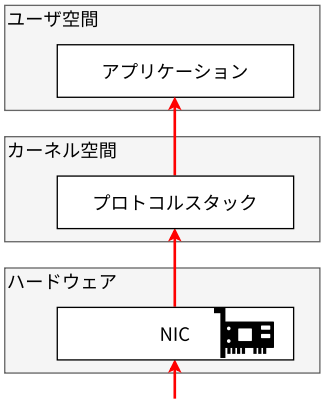

図 1 カーネルによるパケット I/O 処理

3. DPDK(Data Plane Development Kit)

本章では，DPDK によるパケット I/O 処理とその実行モデルについて述べる．また，カーネルによるパケット I/O 処理と DPDK によるパケット I/O 処理の比較や DPDK によるパケット I/O 処理の問題点についても述べる．

3.1 DPDK によるパケット I/O 処理

DPDK [1] とは 2010 年に Intel によって作られたパケット処理を高速化するためのライブラリである．

DPDK は特定の CPU コアを専有することによって，NIC を常時ポーリングで監視する．そのため，DPDK によるパケット I/O 処理（図 2）では，一定時間に受信するパケットの量が増えても，コンテキストスイッチが増加することはない．

また，DPDK によるパケット I/O 処理では，NIC は受信したパケットをユーザ空間からアクセス可能な主記憶領域に書き込む．そのため，受信したパケットをカーネル空間からユーザ空間にコピーしなくても，ユーザ空間のアプリケーションがパケットにアクセスすることができる．

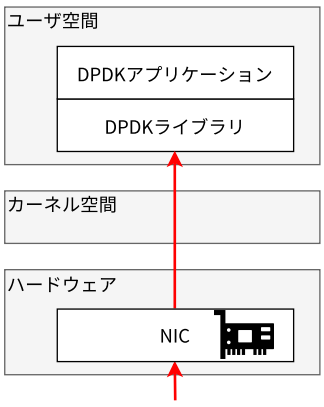

図 2 DPDK によるパケット I/O 処理

3.2 DPDK の実行モデル

DPDK の実行モデルには Run-to-Completion モデルと Pipeline モデルの二つがある．Run-to-Completion モデルは受信処理，パケット処理，送信処理を一つの論理コアで行うモデルである（図 3）．パケット処理が重い場合は受信処理に CPU リソースが割り当たらず，パケットロスが生じるため，パケット処理ではパケットヘッダの書き換えといった軽い処理が一般的には行われる．Pipeline モデルは受信処理，パケット処理，送信処理をそれぞれ別の論理コアで行うモデルである（図 4）．受信処理を行う論理コアとパケット処理を行う論理コアが別であるため，パケット処理が重い場合でもパケットロスが生じることはない．しかし，それぞれの処理が論理コアを専有するため，CPU リソースやパケット処理時に必要なデータの大部分が L1 キャッシュに存在し，結果として処理速度が上がるという効果を有効活用できない．

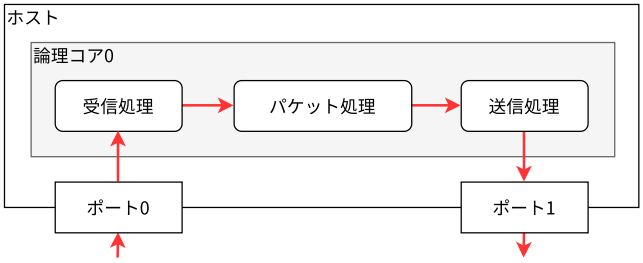

図 3　Run-to-Completion モデル

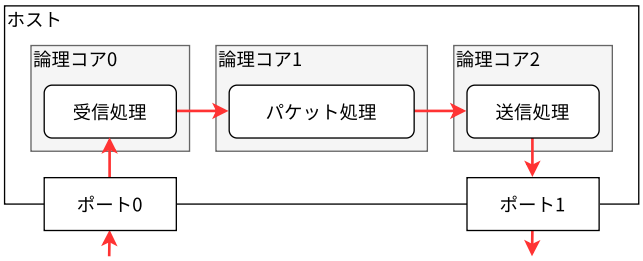

図 4　Pipeline モデル

3.3 パケット I/O 処理の比較

文献 [2] で調査された，Linux4.18 のカーネルによるパケット I/O 処理と DPDK によるパケット I/O 処理の通信スループットを図 5 に示す．このグラフの横軸は使用した CPU コアの数，縦軸は受信したパケットをすべてドロップしたときのスループットを表している．また，緑は DPDK によるパケット I/O 処理，ピンクはカーネルによるパケット I/O 処理の結果である．グラフより，DPDK のスループットはカーネルのスループットに比べて最大 8 倍高いことが確認できる．

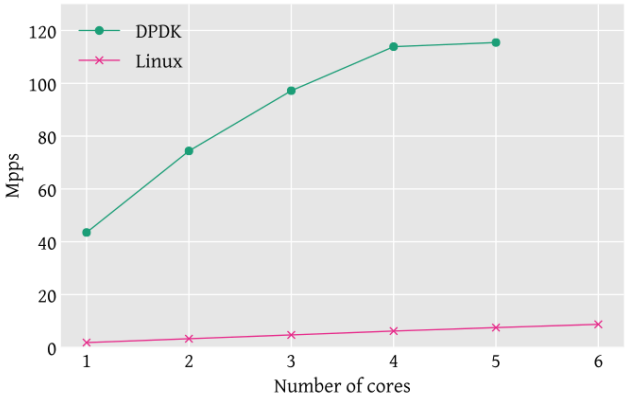

図 5　パケット I/O 処理の比較

3.4 DPDK によるパケット I/O 処理の問題

DPDK は特定の CPU コアを専有することによって，NIC を常時ポーリングで監視するため，通信負荷が低いときでも CPU リソースを無駄に使用する．文献 [2] で調査された，カーネルによるパケット I/O 処理と DPDK によるパケット I/O 処理の CPU 使用率を図 6 に示す．このグラフの横軸は通信負荷，縦軸は CPU 使用率を表している．また，緑は DPDK によるパケット I/O 処理，青はカーネルによるパケット I/O 処理の結果である．グラフから，カーネルによるパケット I/O 処理の CPU 使用率は通信スループットが高くなるにしたがって徐々に増えていくことが確認できる．それに対して，DPDK によるパケット I/O 処理の CPU 使用率は通信スループットにかかわらず常に 100% であることが確認できる．

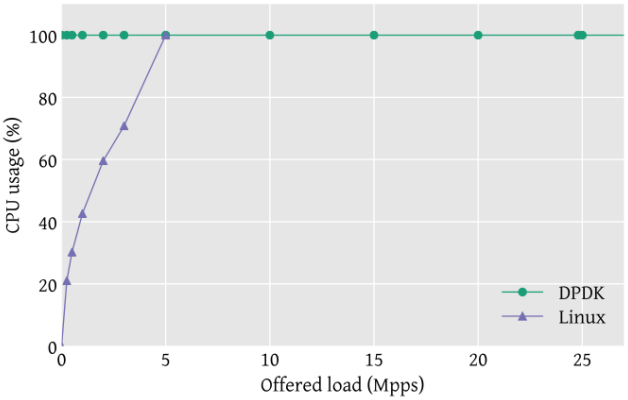

図 6　DPDK によるパケット I/O 処理の問題

4. 提案手法

本研究は DPDK の Run-to-Completion モデルを用いた L2 分散計算環境を提案する．本章では，その概要として DPDK による L2 通信を用いることと Run-to-Completion モデルを採用することについて述べる．

4.1 DPDK による L2 通信の使用

第 2 章で述べたとおり，TCP/IP による通信ではルーティング，誤り制御，順序制御といった制御が行われる．しかし，第 1 章で述べた本研究が提案する分散計算環境の前提を踏まえると，これらの制御はオーバーヘッドである．

カーネルによるパケット I/O 処理は，一定時間に受信するパケットの量が増えると，コンテキストスイッチが増加して，割り込み以外の処理時間が相対的に少なくなる．また，ユーザ空間のアプリケーションがパケットにアクセスするには，受信したパケットをカーネル空間からユーザ空間にコピーしなければならない．そのため，カーネルのパケット I/O 処理は DPDK のパケット I/O 処理に比べて低速である．

そこで，本研究では分散計算環境の計算機間の通信に DPDK による L2 通信を用いる．これによって，TCP/IP によるさまざまな通信処理のオーバーヘッドを低減するとともに，カーネルによるパケット I/O 処理に比べて高速なパケット I/O 処理を用いることができる．

4.2 Run-to-Completion モデルの採用

第 3 章で述べたとおり，DPDK の Pipeline モデルはそれぞれの処理が論理コアを専有するため，CPU リソースや L1 キャッシュを有効活用できない．また，DPDK は特定の CPU コアを専有することによって，NIC を常時ポーリングで監視するため，通信負荷が低いときでも CPU リソースを無駄に使用する．

そこで，本研究では Run-to-Completion モデルを採用する（図 7）．受信処理，パケット処理，送信処理を一つの論理コアで行う Run-to-Completion モデルを用いることで，CPU リソースを有効活用できる．また，本研究が提案する分散計算環境の計算機間でやり取りされるデータのサイズは L2 フレームより小さいため，計算処理時に L1 キャッシュを有効活用できる．さらに，接続された計算機が協調して処理を実行することによって，通信負荷が低い状態にならず，CPU リソースの無駄を削減することができる．

5. 事前実験 1

本章では，DPDK の Run-to-Completion モデルで実行する処理の負荷によって，スループットはどのように変化するかを調査するために行った事前実験 1 について述べる．

5.1 事前実験 1 用のプログラム

事前実験 1 用のプログラムとして，クライアントから送られてきたパケットを一定時間待機させてから送り返すプログラムを作成した（図 8）．DPDK の Run-to-Completion モデルで実行する処理は，受信したパケットを待機させる処理である．パケットを待機させる時間を変更することにより，DPDK の Run-to-Completion モデルで実行する処理の負荷を仮想的に変更することができる．

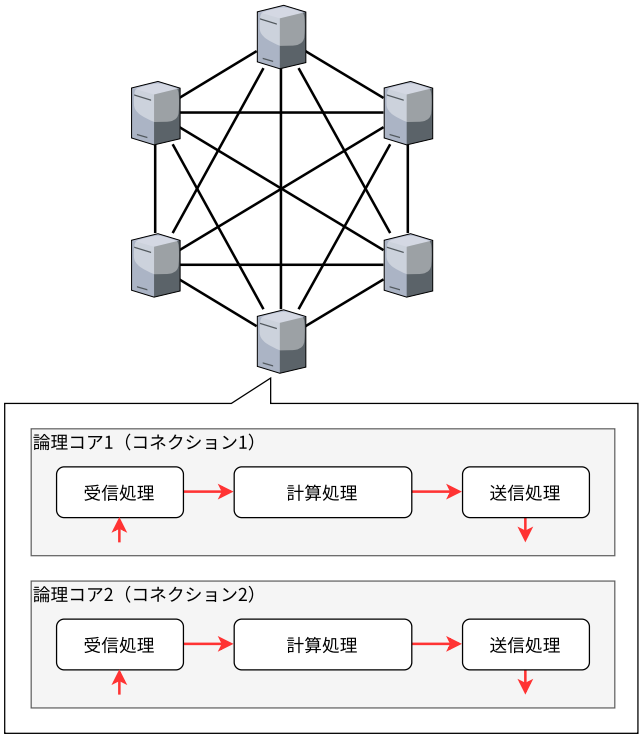

図 7 提案手法

図 8 事前実験 1 用のプログラム

5.2 実験環境

事前実験 1 で用いたネットワーク構成を図 9 に示す．ホスト A においてクライアントを動作させ，ホスト B において事前実験 1 用のプログラムを動作させた．クライアントには，DPDK ベースのパケットジェネレータである Pktgen [3] を用いた．事前実験 1 で用いた計算機の性能を表 1，Pktgen の設定を表 2 に示す．

パケットはクライアントが動作するホスト A のポート 0 から送信され，事前実験 1 用のプログラムが動作するホスト B のポート 0 に到着する．到着したパケットは事前実験 1 用のプログラムによって待機させられてから，ホスト B のポート 1 からホスト A のポート 1 へと送り返される．

5.3 実験結果・考察

事前実験 1 の結果を図 10 に示す．このグラフの横軸は

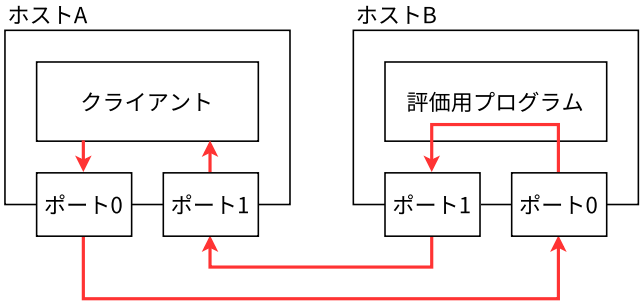

図 9　事前実験 1 で用いたネットワーク構成

表 1　事前実験 1 で用いた計算機の性能

OS	Ubuntu 20.04
CPU	AMD Ryzen 5 3400G (4-cores, 8-threads)
Memory	16GB
NIC	Intel X540-AT2 (10GbE, 2-ports)

表 2　Pktgen の設定

パケットのサイズ	64 バイト, 128 バイト
送信するパケットの数	100,000,000 パケット
送信レート	100%

パケットの待機時間，縦軸は通信のスループットを表している．また，青はパケットサイズが 64 バイトのとき，赤はパケットサイズが 128 バイトのときの結果である．グラフより，パケットサイズが 64 バイトの場合，待機時間が 1600ns 以下のときスループットは一定であり，それを超えると単調減少することを確認した．また，パケットサイズが 128 バイトの場合，待機時間が 3200ns 以下のときスループットは一定であり，それを超えると単調減少することを確認した．よって，送信するパケットのサイズが小さく，送信レートが 100%という厳しい条件でも，処理時間が 1600ns 以下の計算処理であればスループットに影響を与えずに DPDK の送受信スレッドで実行できると考える．

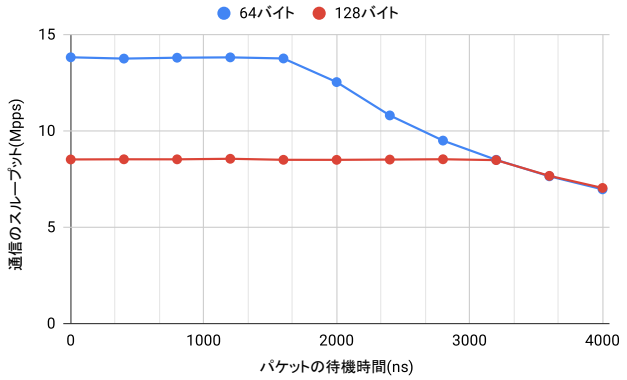

図 10　事前実験 1 の結果

6.　事前実験 2

事前実験 2 では，送られてきた 32 個の値を加算し送り返すプログラムにおいて，TCP/IP による通信を用いる場合と提案手法を用いる場合の単位時間あたりの演算性能を比較した．

6.1　事前実験 2 用のプログラム

事前実験 2 用のプログラムとして，クライアントから送られてきた 32 個の値を加算し送り返すプログラムを作成した（図 11）．DPDK の Run-to-Completion モデルで実行する処理は受信したパケットに含まれる 32 個の値を加算する処理である．このプログラムは TCP/IP による通信を用いるものと提案手法を用いるものの 2 種類を作成した．

図 11　事前実験 2 用のプログラム

6.2　実験環境

事前実験 2 で用いたネットワーク構成は事前実験 1 で用いたもの（図 9）と同じである．ただし，クライアントにはパケットサイズが 64 バイトのパケットを送信レート 100% で送信し続ける自作プログラムを用いた．送信されるパケットのペイロードは要素数が 32 でデータ型が uint16_t の配列である．なお，事前評価 2 で用いた計算機の性能は事前実験 1 で用いたもの（表 1）と同じである．

6.3　実験結果・考察

事前実験 2 の結果を図 12 に示す．このグラフの横軸は内部ループ回数，縦軸は単位時間あたりの演算性能を表している．また，青は TCP/IP による通信を用いた場合，赤は提案手法を用いた場合の結果である．事前実験 2 用のプログラムでは，仮想的に内部演算量を増やすために，同一の加算処理を繰り返しており，その回数が内部ループ回数である．グラフより，提案手法を用いた場合の演算性能は TCP/IP による通信を用いた場合に比べて，内部ループ回数が 1 回の場合は 30 倍，10 回の場合は 4 倍，100 回の場合は 0.4 倍高いことを確認した．この結果より，分散計算環境において，DPDK による L2 通信を用いることと DPDK の Run-to-Completion モデルで処理を行うことは有効であると考える．なお，提案手法を用いた場合の演算性能が内部ループ回数が多くなるにつれて下がっていくのは，計算処理量の増加により受信処理に割り当てられる CPU リソースが相対的に減少したためである．

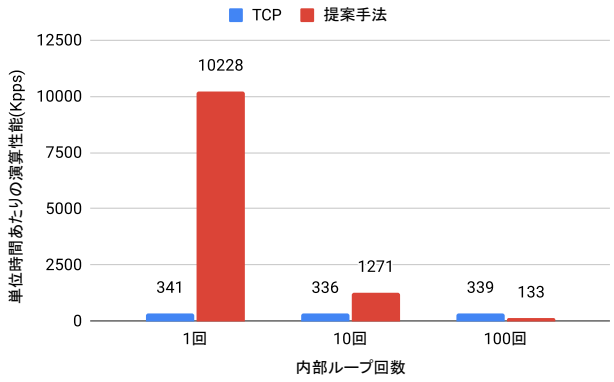

図 12　加算処理の結果

7.　評価

本章では，提案手法の有効性を確認するために行った評価について述べる．評価では，単回帰分析において，1 台での集中学習を行う場合，TCP/IP による通信を用いた分散学習を行う場合，提案手法を用いた分散学習を行う場合の実行時間を比較した．

7.1　評価用プログラム

評価用プログラムとして単回帰分析を行うプログラムを作成した．単回帰分析とは，与えられたデータを用いて $y = ax + b$ の傾き a と切片 b を求める問題である．パラメータの最適化には確率的勾配降下法（Stochastic Gradient Descent, SGD）を用いた．確率的勾配降下法とは，データをランダムに選んで以下の更新を繰り返し，f を最小化するアルゴリズムである．なお，パラメータを w，学習率を α，目的関数を f とする．

$$w \leftarrow w - \alpha \nabla f(w) \tag{1}$$

複数台の計算機で分散して学習を行う場合はパラメータの集約が必要となる．パラメータの集約には Gossip Learning [4], [5] を用いた．Gossip Learning において，各計算機は確率的勾配降下法によってパラメータを更新する．そして，定期的にランダムに選んだ他の計算機と通信し，パラメータの平均を計算する．このようなパラメータの更新を繰り返していくことによって学習が進む．

このプログラムは 1 台での集中学習を行うもの，TCP/IP による通信を用いた分散学習を行うもの，提案手法を用いた分散学習を行うものの 3 種類を作成した．

7.2　評価環境

評価で用いたネットワーク構成を図 13 に示す．計算機を接続し，それぞれの計算機で評価用プログラムを動作させた．なお，評価で用いた計算機の性能は事前実験 1 で用いたもの（表 1）と同じである．

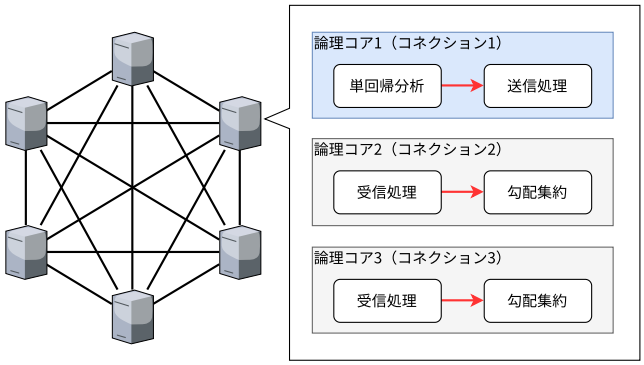

図 13　評価で用いたネットワーク構成

7.3　評価結果・考察

評価結果を図 14 に示す．このグラフの横軸は左から 1 台での集中学習の場合，TCP/IP による通信を用いた 2 台での分散学習の場合，提案手法を用いた 2 台での分散学習の場合を表しており，縦軸は実行時間を表している．グラフより，提案手法を用いた 2 台での分散学習の実行時間は，1 台での集中学習の場合に比べて 2 倍，TCP/IP による通信を用いた 2 台での分散学習の場合に比べて 1.2 倍高速であることを確認した．よって，分散計算環境において，DPDK による L2 通信を用いることと DPDK の Run-to-Completion モデルで処理を行うことは有効であると考える．

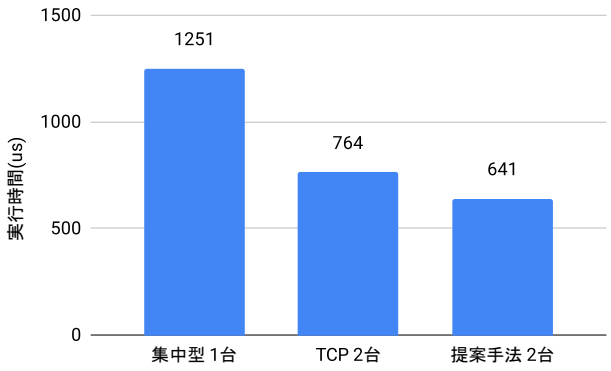

図 14　評価結果

8.　関連研究

多元連立一次方程式の緩和法解析の分散処理に DPDK による通信を用いる研究 [6] がある．1992 年に小石らによって行われた多元連立一次方程式の緩和法解析の分散処理に関する研究 [7] では，UDP によるブロードキャストが行われていた．そこで，文献 [6] では多元連立一次方程式の緩和法解析の分散処理に DPDK を用いることで，通信オーバーヘッドの削減による計算の高速化を行った．その結果，DPDK を用いた緩和法解析の分散処理アプリケーションは，UDP を用いた分散処理よりも最大 40.1%高速

化された．しかし，文献 [6] では受信スレッド，収束判定または計算のスレッド，送信スレッドのそれぞれが論理コアを使用する Pipeline モデルを用いているため，CPU リソースや L1 キャッシュを有効活用できない．

MPI 通信のデータ転送に DPDK を用いる研究 [8] がある．多くの MPI ライブラリはデータ転送を行うレイヤが独立しており，ユーザの環境に合わせてデータ転送方式やデバイスを MPI プログラムの実行に柔軟に切り替えることができる．そこで，文献 [8] は MPI ライブラリの新たなデータ転送モジュールとして DPDK によるデータ転送モジュールを提案した．通信のスループットが低いときは Run-to-Completion モデル，高いときは Pipeline モデルを使用するようになっている．その結果，TCP/IP ソケットによるデータ転送を用いた場合と比べ，通信遅延を最大 77%改善することができた．しかし，文献 [8] では ACK パケットの授受を実装することによって，パケットロスに対する制御を行っているため，通信のオーバーヘッドがある．

9. まとめと今後の課題

本稿では，DPDK の Run-to-Completion モデルを用いた L2 分散計算環境を提案した．計算機間の通信に DPDK による L2 通信を用いることによって，TCP/IP による通信のオーバーヘッドを低減するとともに，カーネルによるパケット I/O 処理に比べて高速なパケット I/O 処理を用いることができる．また，Run-to-Completion モデルを採用することによって，Pipeline モデルを用いる場合に比べて CPU リソースや L1 キャッシュを有効活用できる．送られてきた 32 個の値を加算し送り返すプログラムを用いた事前実験 2 では，提案手法を用いた場合の処理性能は TCP/IP による通信を用いた場合に比べて，内部ループ回数が 1 回の場合は 30 倍，10 回の場合は 4 倍，100 回の場合は 0.4 倍高いことを確認した．また，単回帰分析を行うプログラムを用いた評価では，提案手法を用いた 2 台での分散学習の実行時間は，1 台での集中学習の場合に比べて 2 倍，TCP/IP による通信を用いた 2 台での分散学習の場合に比べて 1.2 倍高速であることを確認した．

今後の課題としては，使用する計算機の台数を増やして評価を取ること，提案手法のどの部分が性能向上に寄与しているのかを調べるために Raw Socket を用いてカーネルによる L2 通信を実装して評価を取ること，ロジスティック回帰やサポートベクターマシン（Support Vector Machine, SVM）といった複雑な機械学習を実行することなどがある．

参考文献

[1] The Linux Foundation: Home - DPDK, The Linux Foundation (online), available from ⟨https://www.dpdk.org/⟩ (accessed 2022-06-10).
[2] Toke Høiland-Jørgensen and Jesper Dangaard Brouer and Daniel Borkmann and John Fastabend and Tom Herbert and David Ahern and David Miller: The eXpress data path: fast programmable packet processing in the operating system kernel, *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, Association for Computing Machinery, pp. 54–66 (2018).
[3] Keith Wiles: pktgen/Pktgen-DPDK: DPDK based packet generator, Intel Corporation (online), available from ⟨https://github.com/pktgen/Pktgen-DPDK⟩ (accessed 2022-06-10).
[4] Peter H Jin and Qiaochu Yuan and Forrest Iandola and Kurt Keutzer: How to scale distributed deep learning?, *arXiv preprint arXiv:1611.04581* (2016).
[5] 小国英明，高橋良希，首藤一幸：広域分散を想定した深層学習手法の比較，データ工学と情報マネジメントに関するフォーラム (2019).
[6] 伴野隼一，矢吹道郎：DPDK の超高速通信を活用した，緩和法解析の分散処理に関する研究，研究報告マルチメディア通信と分散処理（DPS），Vol. 2021-DPS-186, No. 15, pp. 1–7 (2021).
[7] 小石　昇：分散処理におけるブロードキャストを利用した共有データの参照および更新に関する研究，修士論文，上智大学理工学研究科・電気電子工学専攻 (1992).
[8] 島田明男，早坂光雄：DPDK によるデータ転送を用いた MPI 通信の実装，研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2017-HPC-158, No. 13, pp. 1–7 (2017).