特別研究報告

題目

リソース分離型マイクロデータセンターにおける リモートメモリ活用手法の性能評価と考察

指導教員
村田 正幸 教授

報告者
生駒昭繁

2020 年 2 月 10 日

大阪大学 基礎工学部 情報科学科

令和元年度 特別研究報告

リソース分離型マイクロデータセンターにおける リモートメモリ活用手法の性能評価と考察

生駒昭繁

内容梗概

近年、計算機および計算機によって構成されるネットワークやインターネットの発展は著しく、手元の機器では実行困難な処理も、クラウドを用いることでネットワークにさえつながっていれば実行することができるようになった。クラウドによる処理は、ユーザ側から離れた場所にある大規模なデータセンターとの通信を行い、データセンター側で処理をしていく。そのため、データセンターとの通信による遅延が発生し、リアルタイム性が求められる処理には向いていない。そこで、ユーザにより近いエッジに小規模のデータセンターを置き、そこで様々な処理を行えるようにするマイクロデータセンターが提案されている。マイクロデータセンターは、大規模なデータセンターと比べて保有しているリソースの量は限られているため、柔軟なリソース管理による無駄のないリソースの使用が重要となる。これを実現するために、サーバ単位で構成されていたデータセンターを CPU やメモリ、ストレージのようなリソースごとに分解し、リソース単位で構成するリソース分離型のデータセンターの技術をマイクロデータセンターに用いることが提案されている。この技術の導入によって、リソース単位での最適化が行え、利用サービスごとのきめ細かいリソース配分や、リソース管理コストの低下が期待される。また、リソース分離したメモリであるリモートメモリは、サーバ単位ではなく処理単位でメモリを割り当てていくことが可能なため、メモリの使用率を上げ、より効率的なメモリの使用を可能とする。しかし、リソース分離型のデータセンターはリソース間でのやり取りをネットワークによって行うため、リソース間通信による遅延が発生し、性能が低下してしまう恐れがある。特に、メモリと CPU 間の通信による遅延は要件が厳しいため、性能の低下の直接的な原因となってしまう。リソース分離型のマイクロデータセンターの利点を最大限に生かしていくためには、リモートメモリと CPU 間の遅延による影響を最小限に抑えなければならない。そこで、リモートメモリとの通信遅延による性能の低下を抑えられるような帯域幅やレイテンシを求め、実際にリソース分離型マイクロデータセンターを用いるにあたってのリソースの配置方法や分配方法について考察を行う。そのために、プログラムの実行時に性能低下の直接的な原因となるメモリと CPU

の通信によるレイテンシ、帯域幅を変化させていき、それぞれメモリと CPU が分離されていないときと比べてどれだけ性能に影響があるかを測定し、性能低下を抑えられる最低限の帯域幅とレイテンシを求めた。その結果、帯域幅は 40Gbps あれば十分であることが示された。また、実行する処理ごとにレイテンシの要件は異なるため、それらに対応するための適切なリモートメモリの配分や配置についての考察を行い、少なくともリソースから 400m 以内にはリモートメモリを配置する必要があり、リモートメモリと CPU との距離やリモートメモリの容量を加味した、リモートメモリの適切な資源配分アルゴリズムが必要であるとわかった。

主な用語

マイクロデータセンター、リモートメモリ、レイテンシ、帯域幅、リソース分離、行列計算、機械学習

目 次

図 目 次

表 目 次

1　はじめに

計算機および計算機によって構成されるネットワークやインターネットの発展は著しく、ユーザはクラウドに接続することでユーザ端末では実行困難である大規模な計算や、大規模なストレージのような様々なサービスを手軽に利用することができる。さらに、IoT 機器の普及によって大量の多種多様なデータがクラウド上で処理されるようにもなった。しかし、クラウドでの処理は端末からデータセンターに接続するまでにかかってしまう通信遅延や、処理したいデータをインターネットを介してクラウドに上げることへのセキュリティ上の問題、IoT の発達によって爆発的に増えているデータとクラウドの通信によるトラヒック量の増加という問題がある。これらの問題に対応するために、ユーザからより近い場所やデータの収集装置の近くにリソースを配置し、処理を行えるようにするエッジコンピューティングの研究開発が進められている。これによって、処理の低遅延化やトラヒック量の増加への対応が期待されている [1]。その中で、ユーザからより近い場所やデータの収集装置の近くにデータセンター自体を置き、そこでデータの処理を行うというマイクロデータセンターが考えられている [2]。

現在主に利用されているデータセンターは、CPU やメモリ、ストレージのようなさまざまなリソースが緊密に統合して成り立っているサーバを中心として構築されている。しかし、リソースが統合されているため、新たなリソースを追加することや、リソースごとのきめ細かい管理をするのが難しいといえる。マイクロデータセンターは、大規模なデータセンターと比べて保有しているリソースの量は限られているため、柔軟なリソース管理による無駄のないリソースの使用が重要となる。これらの事象を実現するための方法として、リソース分離型のデータセンターの技術が考えられる。リソース分離型のデータセンターは、図 1 に示すようにデータセンター内でサーバ単位で集約されていた CPU やメモリのようなリソースをリソースの機能ごとにリソースブレードとして分解し、データセンター内の構成をリソースブレードごとに分離させたものである [3]。リソース分離型のデータセンターは、実行処理ごとに分離されたリソースそれぞれを独立して割り当てることができるため、リソースの使用の最適化を行うことが可能である。これによって、データセンター内のリソース使用率を向上させることができる [4]。さらに、リソース単位で独立しているため、リソースの更新を他のリソースとのつながりを気にせずに自由に行うこともできる。

しかし、リソース単位でデータセンターを構成するため、リソースブレード間でのデータの転送時に発生する遅延による性能低下の影響が問題となっている。そこで、性能の低下を抑えるために必要な通信要件の研究が進められており、RDMA の利用による高速化 [5] や光インターコネクトの利用による高速化 [6] が提案されている。しかし、これらの研究は分散処理のような大規模な並列演算処理を対象としており、マイクロデータセンターで動作させ

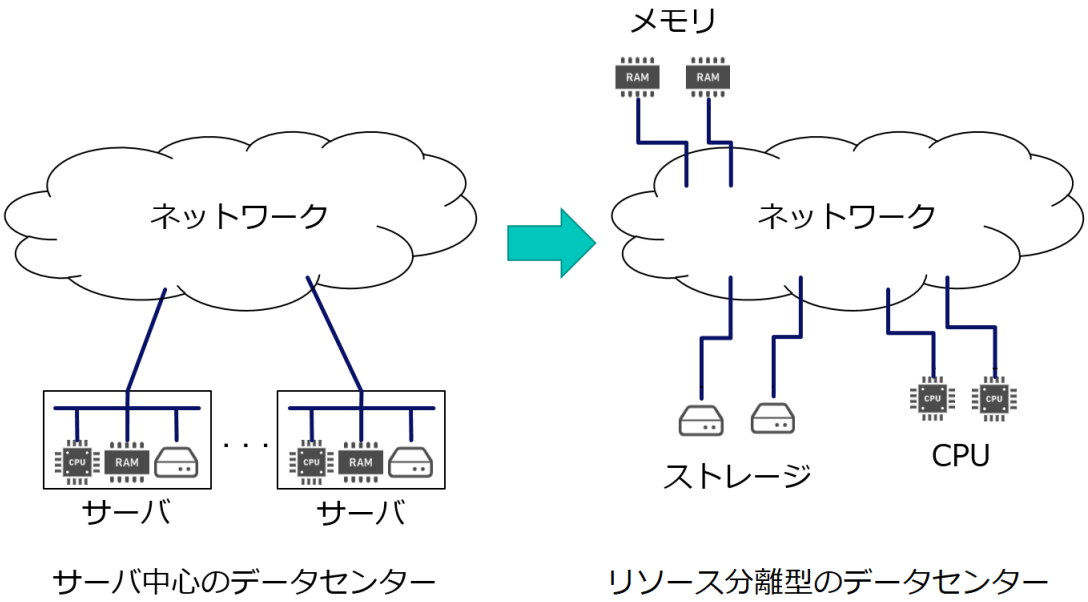


図 1: サーバ中心のデータセンターとリソース分離型のデータセンター

るような画像認識等のアプリケーションを想定した検討は行われていない。さらに、これらの研究で対象とされていた大規模なデータセンターと異なり、マイクロデータセンターではリソース間の距離が小さく、発生しうる遅延も小さいと考えられる。

そこで、本研究報告では、マイクロデータセンターに焦点を当て、マイクロデータセンターにおいてリソース分離型の構成を構築する際の要件について議論を行う。CPU とリソース分離を行ったメモリであるリモートメモリ間の通信については要求されるレイテンシと帯域幅の要件が厳しく、性能低下の直接的な原因となっている [5]。リモートメモリはメモリの使用率を上げ、より効率的なメモリの使用を可能にする技術である [7] が、この利点を最大限に生かすためには、リモートメモリの活用による性能低下を抑える必要がある。そこで、マイクロデータセンターでの実行が見込まれる処理において CPU とリモートメモリの通信による遅延が性能にどれほど影響するのかを調査し、実際にリソース分離型のマイクロデータセンターを利用していく際に必要とされる帯域幅とレイテンシの要件、実際に利用していく際のリソースの配置やリモートメモリの分配方法について考察する。

本研究報告の構成は以下の通りである。2 章ではリソース分離型のデータセンターにおけるネットワーク要件についての関連研究について述べる。3 章では本報告で想定しているリソース分離型のマイクロデータセンターについて説明する。4 章でリモートメモリの活用手法における性能評価と考察を行う。最後に、5 章で結論と今後の課題について述べる。

2　関連研究

本章では、本報告の研究に関するリソース分離型のデータセンターとそのネットワーク要件についての研究について説明する。

リソース分離型データセンターとは、CPU、メモリ、ストレージ、通信機器のようなリソースを集約したサーバの集合として構成されていたデータセンターをリソースごとに分離させ、データセンターをリソースの集合として構成したものである。これによって、リソースごとの最適化を行え、リソースの効率的な利用とそれによる性能の向上が期待される [3]。ただし、既存の OS やソフトウェアシステムではリソース分離型のデータセンターを適切に管理することはできない。そこで、リソース分離型データセンターを実現するためのシステムについての研究 [8] やリソース分離型データセンター対応した OS モデルの研究 [9] のような研究が進められている。

しかし、リソース分離型のデータセンターでは、リソースごとの通信遅延のために発生する遅延のため従来のサーバ中心のデータセンターと比べ性能が下がってしまうおそれがある。文献 [5] では、リソース分離型のデータセンターにおいて、従来のサーバ中心のデータセンターで処理を行う時と同様の性能で実行できるような帯域幅とレイテンシの要件、キャッシュメモリ容量、それを実現するための方法について述べている。このとき、文献 [5] では、帯域幅とレイテンシの要件についての評価のために、Hadoop、Spark、GraphLab、Timely dataflow 、Spark Streaming、memcached、HERD、SparkSQL といったデータセンターで用いられる分散処理プログラムを用いている。

研究の結果として、リソース分離型のデータセンターで分散処理プログラムをサーバ中心のデータセンターと同様の性能で実行するには、レイテンシを 3μs から 5μs に抑えることが必要で、帯域幅としては 40Gbps 以上が必要だということが示されている。また、リソースごとに分離しないで CPU に直接取り付けられるキャッシュメモリの容量としては、全体で使用するメモリの 20% 程度が必要であると示されている。ただし、帯域幅やローカルメモリの要件については既存の製品をそのまま使用しても実現可能であるのに対し、レイテンシについての要件が実現困難であることが言及されている。文献では、解決策として OS を通さずにデータを転送する RDMA があげられており、これにより OS との通信による遅延がなくなり、一部のアプリケーションではあるが、レイテンシを要件を満たせるようになる。

また、文献 [6] では、リソース分離による性能低下への対応として、光インターコネクトや光スイッチのような光通信技術の利用を提案している。さらに、リソースの設置面積や消費電力の観点からも光通信技術の有用性が示されている。

しかしながら、文献 [5] や文献 [6] では、大規模なデータセンターで実行されるような処理を対象としており、マイクロデータセンターで動作させるような画像認識等のアプリケー

ションを想定した検討は行われていない。リソース間でやり取りするデータ量は動作させるアプリケーションによって様々であり、評価するアプリケーションによって必要な通信要件も異なると考えられる。また、文献 [5] では、大規模なデータセンターにおけるリソース分離について考えているが、マイクロデータセンターではリソース間の距離が小さく、発生しうる遅延も小さいと考えられ、リソース分離型の大規模データセンターでは実行が困難であったようなアプリケーションであっても、リソース分離型のマイクロデータセンターでは実行可能である可能性もある。そこで、本報告ではマイクロデータセンターでの実行が想定される機械学習による推論処理についてリソース分離の影響を調査し、必要なネットワーク要件や実際にどの範囲でリソースを配置すればよいのかについて考察を行う。

3　リソース分離型のマイクロデータセンター

本報告では、リモートメモリを用いたリソース分離型のマイクロデータセンターを想定している。そこで、本報告で想定しているリソース分離型のマイクロデータセンターの構造とリモートメモリについての説明を本章で行う。

3.1　リソース分離型のマイクロデータセンターの構造

リソース分離型マイクロデータセンターは、リソース同士が密接につながれ構成されていたマイクロデータセンターをリソースごとに分離させて構成したものである。ここで、リソースとは、CPU、ストレージ、メモリのような計算機の構成要素を指している。各リソースはネットワークによって接続されていて、相互に通信することが可能である。また、リソース単位でデータセンターを構成するため、異なる場所にあるリソースを自身のマイクロデータセンターで行う処理に割り当てることも可能である。ここで、本報告で想定しているリソース分離型マイクロデータセンターの構造を図 2 に示す。

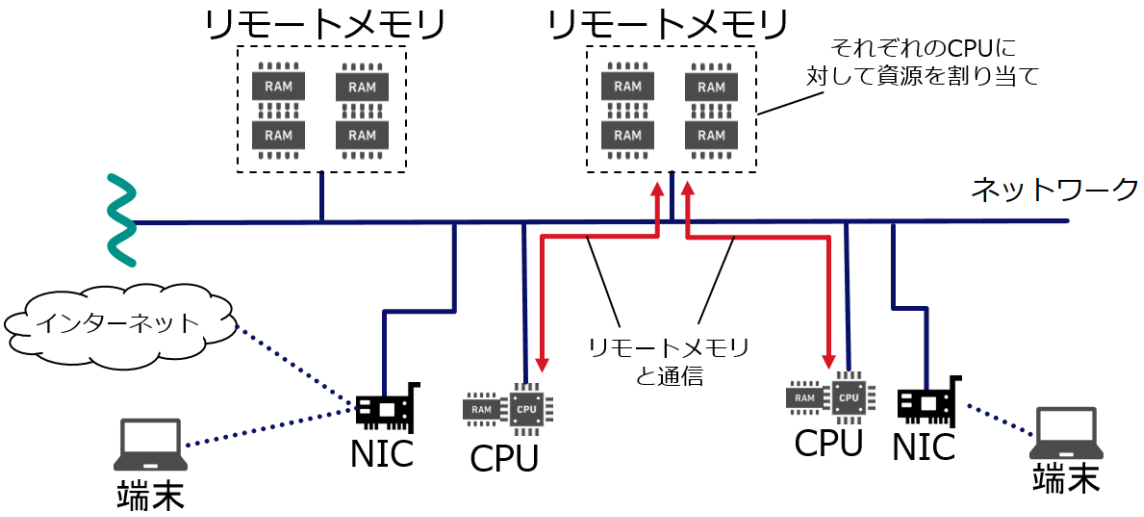


図 2: リソース分離型マイクロデータセンターの構造

図 2 に示すように、リソース同士はネットワークを通してつながれており、各リソースからほかのリソースへの通信はネットワークを通して行われる。このとき、リソース同士の距離が遠すぎるとリソース間の通信による遅延による影響が大きくなるので、リソース同士は一定の範囲内に配置される。図 2 では、異なる CPU が同じリモートメモリと通信しているが、それぞれのリソースが分離しているため、どのリソースであるか関係なく様々なリソースに対して自分の資源を割り当てていくことが可能となっている。他の端末やクラウドと通

信しても問題のない処理であれば、NIC を通してデータをやり取りすることも可能である。また、リソース同士はネットワークでつながれているもののリソース同士が統合しているわけではないため、リソースの取り換えや新しいリソースへの変更を全体の構成を設計し直すことなく行うことが可能となり、導入のコストや管理コストを削減できると考えられる。

3.2 リモートメモリ

リモートメモリとは、ネットワークを経由してアクセス可能なメモリである。リモートメモリは複数のリソースによって共有し、各リソースの処理ごとに必要なサイズのリモートメモリを割り当てるなど、柔軟なメモリ割当が可能となる。サーバ中心のデータセンターでは、サーバに取り付けられたメモリ量に反して実際に使用するメモリ量は少なく、メモリ使用率が低くなる場合があるという問題がある。リソース分離型のマイクロデータセンターで用いられるリモートメモリは、様々な処理に対して必要な分を割り当てていくため、効率的なメモリの利用が可能となる。さらに、実行される処理の増加に対しても、サーバ中心のデータセンターのようにサーバ単位で考えていくのではなく、リモートメモリの容量のみを考えて割り当てていくことができるため、より柔軟な対応が可能となる。

4　リモートメモリを使用した計算機における性能の評価

本章では、リモートメモリを使用した計算機において、実行性能がどれだけ低下するのかを評価し、実際に運用していくにあたっての効率的な利用方法や性能低下を抑えるためのネットワーク要件について考察する。

4.1　評価環境

本報告の性能評価のために、リモートメモリと CPU との通信をシミュレーションするためのプログラムを実装する。CPU とリモートメモリとの通信のシミュレーションのために、文献 [5] で用いられた方法を採用した。この方法では、CPU とメモリとのやり取りを、CPU に取り付けられたローカルメモリと CPU からは離れたところに配置されたリモートメモリとのやり取りに置き換えることで CPU とメモリとのやり取りを再現する。このとき、ローカルメモリは計算機を動作させるのに最低限必要なキャッシュメモリとして働き、本報告では容量を 200MB に固定する。また、リモートメモリとローカルメモリのデータのやり取りはページ単位で行われる。

方法としては、リモートメモリとローカルメモリとのやり取りを行うためのカーネルモジュールをスワップデバイスという形で計算機に導入し、そのカーネルモジュールによって計算機に元から搭載されていたメモリを最低限のキャッシュメモリとして使用するために CPU に取り付けられたローカルメモリと、スワップ領域として振る舞うリモートメモリに分割することで再現する。そして、ページフォルトが発生した場合は、図 3 に示すように、リモートメモリからローカルメモリへの書き込み処理を行うと同時に、カーネルモジュール内で通信によるレイテンシと帯域幅による遅延を挿入する。このように、カーネルモジュールでリモートメモリとメモリのやり取りを管理し、データ転送時に遅延を挿入することで、ローカルメモリとリモートメモリとのやり取りを再現している。ただし、カーネルモジュールはスワップデバイスとして計算機に導入するが、リモートメモリからローカルメモリへの読み出し、ローカルメモリからリモートメモリへの書き込みのみを行う。

ローカルメモリとリモートメモリのやり取りをする際、前述したカーネルモジュールを通してやり取りを行い、そこでパラメータによって設定されたレイテンシ (μs) と帯域幅 (Gbps) をもとにリモートメモリの使用による遅延を挿入する。導入したカーネルモジュールはリモートメモリへの書き込み要求、ローカルメモリからの読み込み要求を受け取ると、通信相手からの応答を待つまでの時間として、レイテンシによる遅延を挿入する。レイテンシによる遅延は設定したナノ秒単位のレイテンシ分の時間がたつまで処理を待機させることで行う。その後、リモートメモリへの書き込み、リモートメモリからの読み込みの処理を行う。

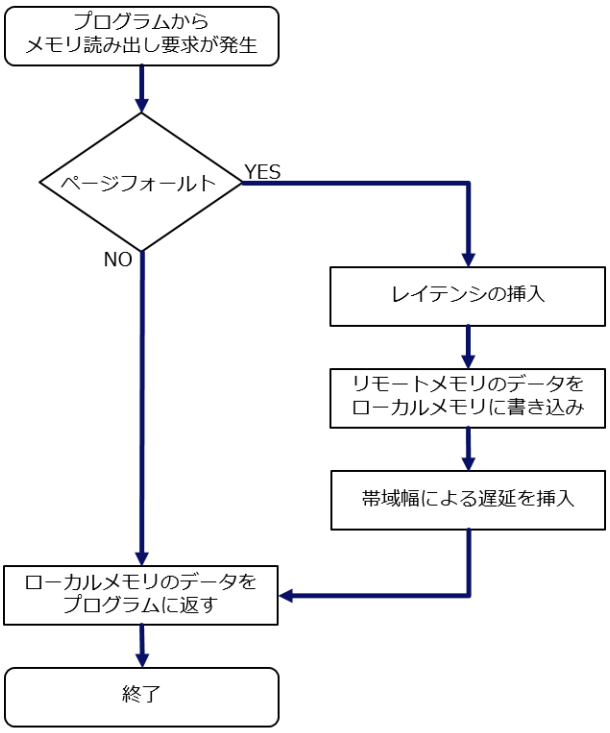


図 3: メモリ読み出し要求の発生時の動作

ここで、データの転送による遅延を設定した帯域幅をもとに処理を待機させることで挿入する。

また、帯域幅による遅延は実際にデータを転送する際に挿入する。本実験環境では、ページサイズは 4KB に設定し、データの転送は 1 ページずつ行われる。そのため、4KB 分のデータを転送する時間を帯域幅による遅延として挿入する。ここで、帯域幅による遅延時間の式を式 (1) に示す。このとき、設定した帯域幅 (bps) を bandwidth、帯域幅による遅延時間 (ns) を $T_{transmit}$ とする。

$$T_{transmit} = \frac{4096 \times\ 8 \times\ 1000000000}{bandwidth} \tag{1}$$

これらの処理によって遅延を挿入することで、CPU とリモートメモリが分離されていて、やり取りに通信遅延が発生する場合の性能評価が可能となる。

本報告では、上記のプログラムを用い、CPU として Intel(R) Xeon(R) CPU E5-2687W 0 @ 3.10GHz を、メモリとして DDR3 SDRAM 持つ計算機を用いて、CentOS7.7 上で評価を行った。なお、本計算機の CPU のバススピードは 8GT/s であり、使用したメモリの転送速度は 12.8GB/s である。

4.2 評価対象

本報告の性能評価の評価対象として、行列計算とその行列計算をもとにしていてかつ、エッジでの利用が見込まれる機械学習による画像分類の処理を採用した。

4.2.1 行列計算

性能の評価のために、比較的大きい行列に対しての畳み込み計算を行う。処理内容としては、4096 × 4096 のランダムな値を格納した行列に対して 3 × 3 の行列を用いた畳み込み計算である。この処理を 20 回繰り返したものを性能評価用のプログラムとする。

4.2.2 画像分類

本報告では、リソース分離型のマイクロデータセンターをエッジで使用することを想定している。そこで、4.2.1 章で説明した行列計算処理を利用し、エッジでの利用も想定される畳み込みニューラルネットワークのモデルを用いた画像分類の処理についても性能評価を行う。画像分類は、MobileNetV2[10]、ResNet[11]、NASNet[12]、Inception-v3[13] のそれぞれの機械学習モデルを用いて行われる。また、機械学習ライブラリとして Tensorflow[14] と Tensorflow のモバイル、エッジ端末向けライブラリである Tensorflow Lite[15] の二つの機械学習ライブラリを用いる。ここで、使用する機械学習モデルとそのサイズを表 1 に示す。表 1 に示したモデルはすべて画像分類のためのモデルであり、畳み込みニューラルネットワークのモデルとなっている。また、Tensorflow で動作させるモデルは Keras[16] によって提供されている学習済みのモデルであり、Tensorflow Lite で動作させるモデルは、Keras によって提供されている学習済みモデルを Tensorflow Lite で動作できるように変換したものである。

プログラムとしては、モデルの読み込み、画像の読み込みと変換、読み込んだ画像に対する推論処理を行う。このとき、バッチサイズは 1 として、異なる 100 個の画像に対して推論処理を行っていく。

表 1: TensorFlow と TensorFlow Lite で動作させるモデルとサイズ

モデル	TensorFlow でのサイズ	TensorFlowLite でのモデル
ResNet50	99MB	98MB
ResNet101	172MB	170MB
ResNet152	232MB	230MB
NASNetLarge	343MB	339MB
NASNetMobile	23MB	21MB
MobileNetV2	14MB	14MB
InceptionV3	92MB	91MB

4.3 評価方法

本報告では、リモートメモリとの通信による遅延を挿入していないときと比べてプログラムの実行時間が相対的にどれだけ長くなったのかを性能低下率として定める。性能低下率は、式 (2) によって求められる。このとき、P は性能低下率、t_{base} はリソースの分離を行わない場合の実行時間、t_d はリソースの分離を行った場合の実行時間を表す。

$$P = \frac{t_d - t_{base}}{t_{base}} \tag{2}$$

本報告では、まず、行列計算についてリモートメモリの影響を調査し、性能低下率を 5% 以内に抑えられるのに必要な帯域幅とレイテンシを求める。そして、行列計算を多く使う処理である機械学習による画像分類処理を評価し、行列計算の結果も考慮して考察を行う。

また、本報告の機械学習による画像分類処理の評価では、リモートメモリの導入による性能低下を調査するために 3 つの観点で調査を行う。まず、レイテンシが 1、10、20 μs、帯域幅が 10、40、100Gbps の場合についてそれぞれパフォーマンスの低下度を調べ、モデルや使用するライブラリによって性能の低下度合いに違いが出るのかを調べる。このとき、性能低下の度合いによって分類ができるのであれば分類し、以降の実験で分類ごとのレイテンシ、帯域幅の要件を導出する。また、性能低下率と評価対象との関係について考察する。

次に、帯域幅と性能の関係について調査する。本実験環境では、一回の転送処理あたりにやり取りするデータ量は 4KB に限られているため、一定以上の帯域幅を設定すると性能の低下は収まると考えられる。そこで、レイテンシを 5μs に固定した状態で、帯域幅のみを変化させていき、性能の低下が収まる値を求め、その値を帯域幅の要件とする。

最後に、求めた帯域幅を設定した状態で、レイテンシのみを変化させて性能の低下度合いを調査し、性能の低下を 5% 以内に抑えられるような帯域幅とレイテンシの要件を導出する。

4.4 行列計算の評価結果

4.4.1 帯域幅による性能低下率の評価

レイテンシを 5μs に固定した状態で、帯域幅を 10Gbps、20Gbps、30Gbps、40Gbps、60Gbps、80Gbps、100Gbps に変更していき、そのときの性能低下率について評価した結果を図 4 に示す。図の点は 10 回分の計測した実行時間に対する性能低下率を示し、棒グラフは 10 回分の計測した実行時間の中央値に対する性能低下率を示している。

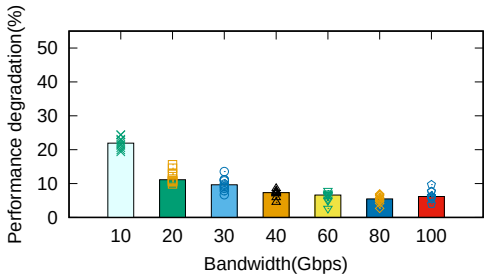
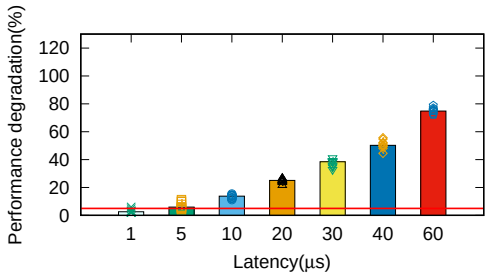


図 4: 行列計算の帯域幅による性能低下率

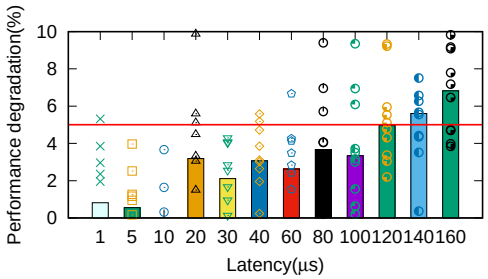
図 4 から、40Gbps 以上の帯域幅で性能低下率の値は横ばいとなっていることが分かる。そのため、40Gbps 以上の帯域幅があっても、性能にはほとんど影響がないということがいえる。以降の評価では、帯域幅を 40Gbps に固定して行う。

4.4.2 レイテンシによる性能低下率の評価

帯域幅を 40Gbps に固定してレイテンシの変更による性能低下率の調査した。このとき、レイテンシを、遅延の影響をほとんど受けないグループでそれ以外のグループで 1μs、5μs、10μs、20μs、30μs、40μs、60μs に変更していき、そのときの性能低下率について評価した。また、行列のサイズによる違いを調査するために行列のサイズを 2048 × 2048 に変更したものに対してもレイテンシを変更していき、性能低下率を評価した。それらの結果を図 5 に示す。図の点は 10 回分の計測した実行時間に対する性能低下率を示し、棒グラフは 10 回分の計測した実行時間の中央値に対する性能低下率を示している。



(a) 4096 × 4096 の行列に対する計算のレイテンシによる性能低下率



(b) 2048 × 2048 の行列に対する行列計算のレイテンシによる性能低下率

図 5: 行列計算のレイテンシによる性能低下率

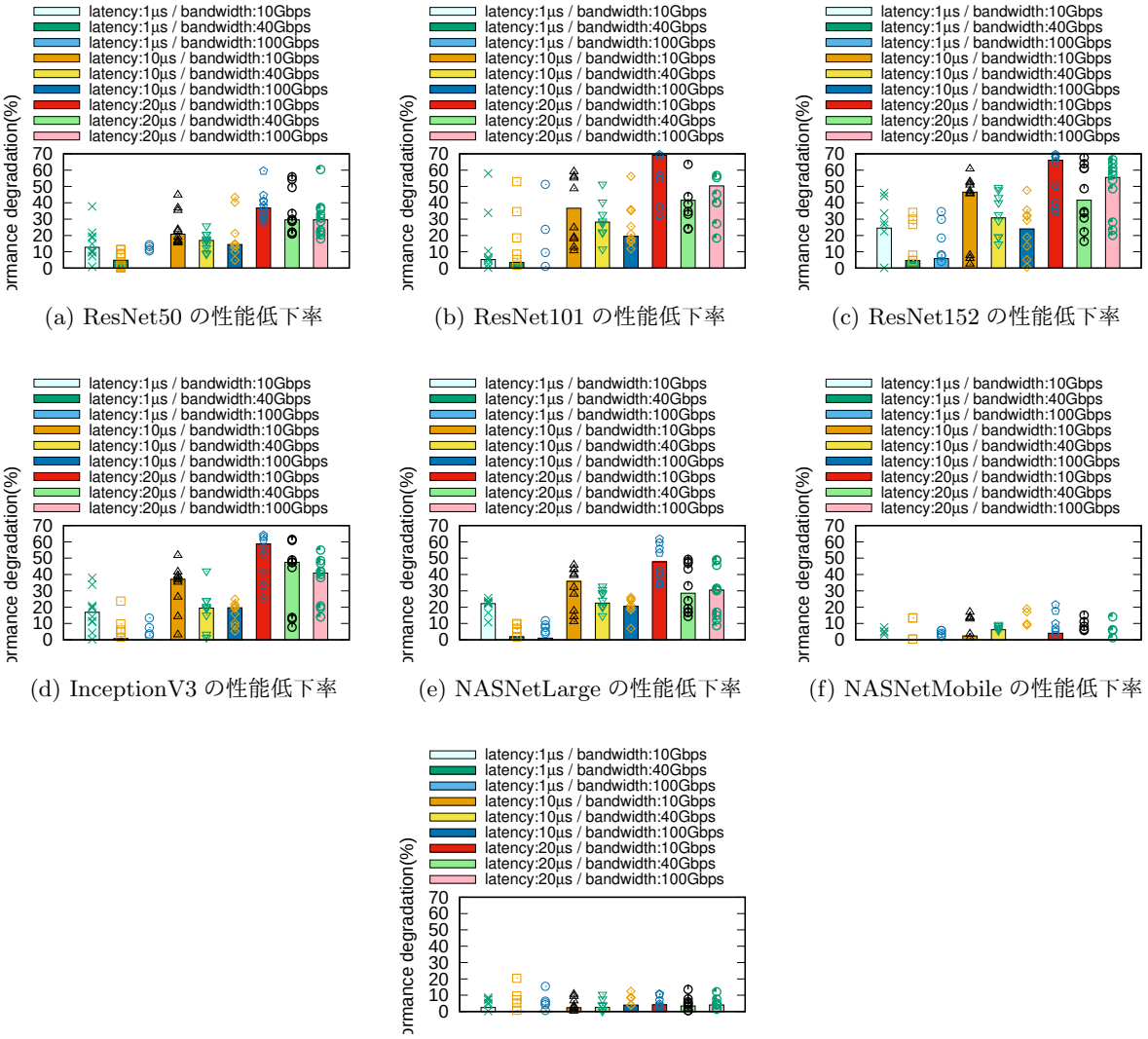
図 5(a) に示すように、4096 × 4096 の行列に対する行列は、計算レイテンシの影響を強く受けていることがわかる。また、レイテンシが 5μs 以上のとき、性能低下率は 5% を上回る結果となった。そのため、行列計算のレイテンシ要件は 1μs といえる。レイテンシ要件が 1μs であるため、離れた場所にあるリモートメモリを使うより、CPU に取り付けられたローカルメモリで実行されるべき処理であるといえる。または、畳み込み計算をはじめとした計算処理を行うための別のリソースを用意し、そこで計算処理を行い、計算結果のみを元のリソースに返すということも考えられる。一方で、図 5(b) から、2048 × 2048 の行列に対する行列計算の性能低下率が 5% を超えるのは、レイテンシが 140μs 以上の時であり、4096 × 4096 の行列に対する行列計算よりも大幅にレイテンシ要件が緩いことがわかる。つまり、行列のサイズによってリモートメモリによる影響は大きく変化するといえる。このことから、行列のサイズによってはリモートメモリを使用しても性能に影響はないといえ、計算用のリソースを用意する場合はローカルで行うべき処理なのか、別のリソースに任せるべき処理なのかを判断するためのアルゴリズムが必要となるといえる。

4.5 画像分類の評価結果

4.5.1 評価対象ごとの性能低下率の評価

レイテンシが 1μs、10μs、20μs で、帯域幅が 10Gbps、40Gbps、100Gbps であるときのすべての場合について、表 1 に示すモデルを用いた画像分類処理による処理を実行し、遅延を挿入しないで実行したときと比較してどれだけ性能が低下したのかを計測する。また、図

6 と図 7 にその結果を示す。図の点は 10 回分の計測した実行時間に対する性能低下率を示し、棒グラフは 10 回分の計測した実行時間の中央値に対する性能低下率を示している。



(a) ResNet50 の性能低下率

(b) ResNet101 の性能低下率

(c) ResNet152 の性能低下率

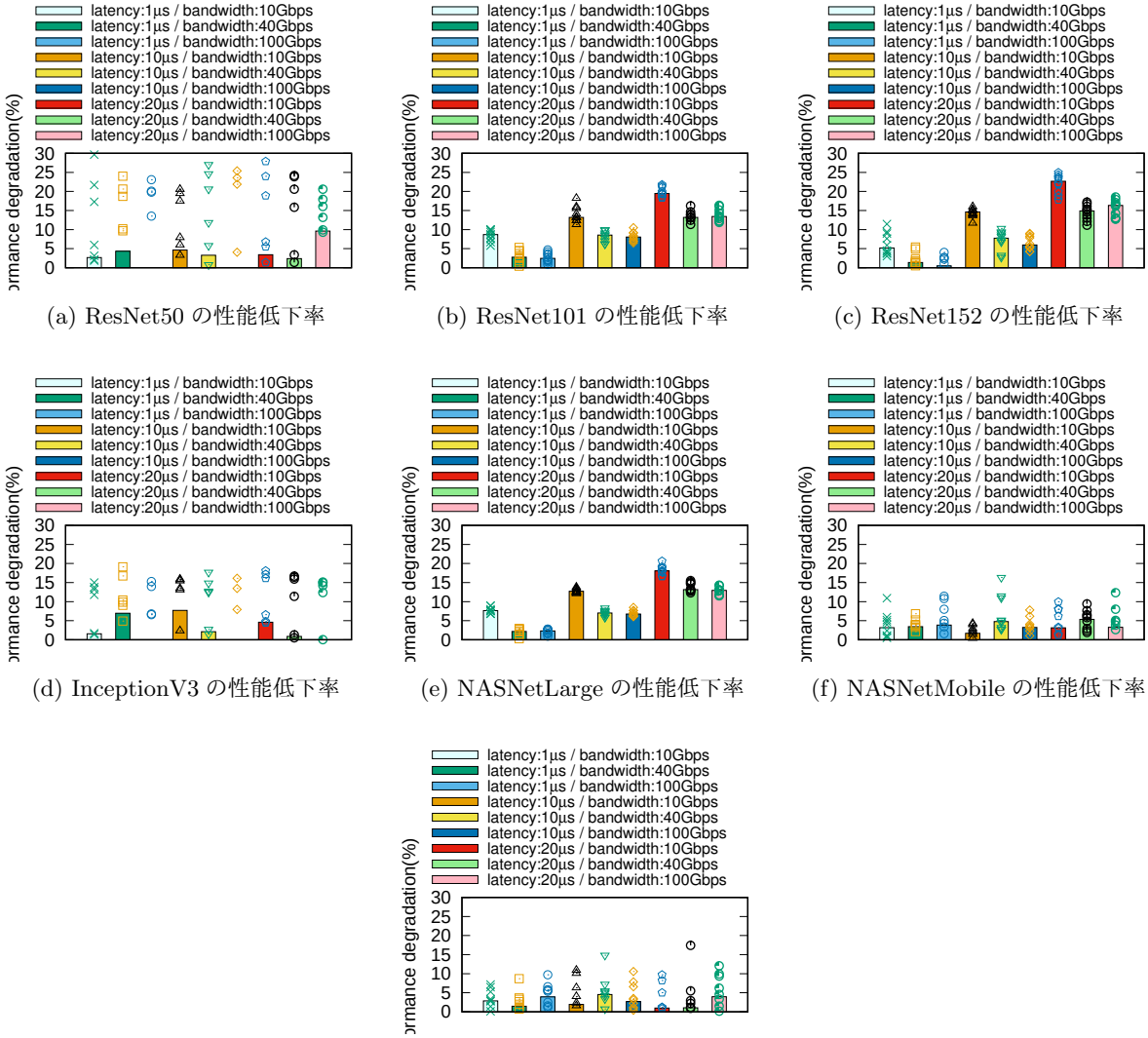
(d) InceptionV3 の性能低下率

(e) NASNetLarge の性能低下率

(f) NASNetMobile の性能低下率

(g) MobileNetV2 の性能低下率

図 6: Tensorflow における評価対象ごとの性能低下率



(a) ResNet50 の性能低下率

(b) ResNet101 の性能低下率

(c) ResNet152 の性能低下率

(d) InceptionV3 の性能低下率

(e) NASNetLarge の性能低下率

(f) NASNetMobile の性能低下率

(g) MobileNetV2 の性能低下率

図 7: Tensorflow Lite における評価対象ごとの性能低下率

図 6 と図 7 に示すグラフから、Tensorflow の MobileNetV2、NASNetMobile と Tensorflow Lite の ResNet50、InceptionV3、NASNetMobile、MobileNetV2 を用いた推論処理において、性能低下率とレイテンシや帯域幅との関係は見られず、性能低下率も 5% を下回ったり、値が安定していないといえ、遅延の挿入による影響をあまり受けていないといえる。一方で、それ以外の評価対象では、帯域幅の値が小さいとき、レイテンシの値が大きい時に性能低下率が大きくなっている傾向がみられ、メモリ分解の影響を大きく受けているといえる。

図 6 と図 7 の結果を踏まえて、遅延の影響をほとんど受けないグループ、遅延の影響で最

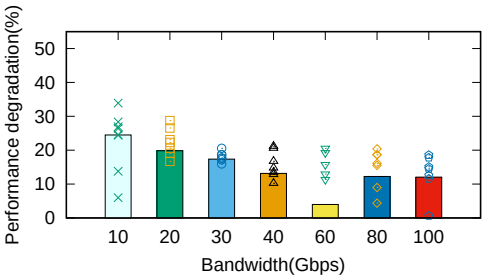
大 20% ほど性能低下がみられるグループ、遅延の影響で最大 30% 以上の性能低下がみられるグループに分類できる。以後の調査では、このグループごとに比較していく。それぞれのグループとの対応を表 2 に示す。

表 2: 各グループと評価対象の対応表

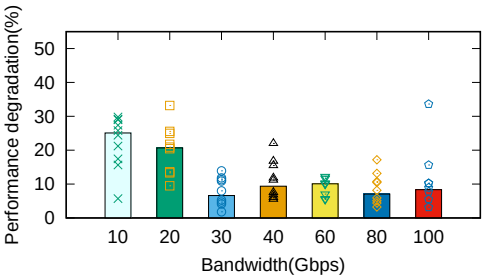
グループ	評価対象名
影響を受けにくいグループ	InceptionV3(Tensorflow Lite) MobileNetV2(Tensorflow) MobileNetV2(Tensorflow Lite) NASNetMobile(Tensorflow) NASNetMobile(Tensorflow Lite) ResNet50(Tensorflow Lite)
最大 20% ほど性能低下がみられるグループ	NASNetLarge(Tensorflow Lite) ResNet50(Tensorflow) ResNet101(Tensorflow Lite) ResNet152(Tensorflow Lite)
最大 30% 以上の性能低下がみられるグループ	NASNetLarge(Tensorflow) InceptionV3(Tensorflow) ResNet101(Tensorflow) ResNet152(Tensorflow)

4.5.2 帯域幅による性能低下率の評価

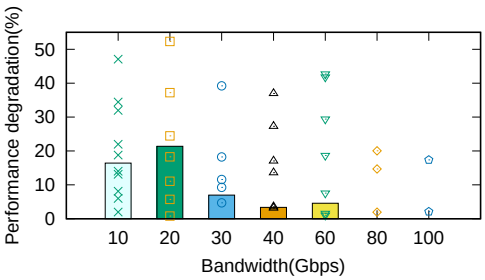
レイテンシを 5μs に固定した状態で、帯域幅を 10Gbps、20Gbps、30Gbps、40Gbps、60Gbps、80Gbps、100Gbps に変更していき、そのときの性能低下率について調査した。図 8 から図 10 に調査結果を示す。また、表 2 に示したグループごとにそれぞれの性能低下率をグラフにしたものを図 11 に示す。図の点は 10 回分の計測した実行時間に対する性能低下率を示し、棒グラフは 10 回分の計測した実行時間の中央値に対する性能低下率を示している。



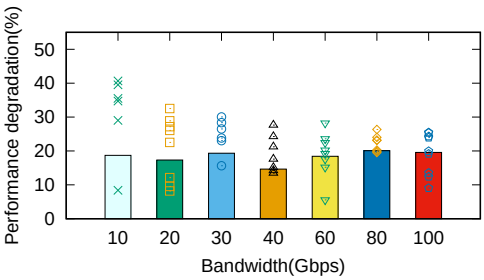
(a) NASNetLarge の帯域幅による性能低下率



(b) InceptionV3 の帯域幅による性能低下率

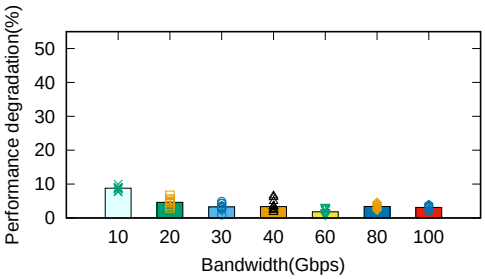


(c) ResNet101 の帯域幅による性能低下率

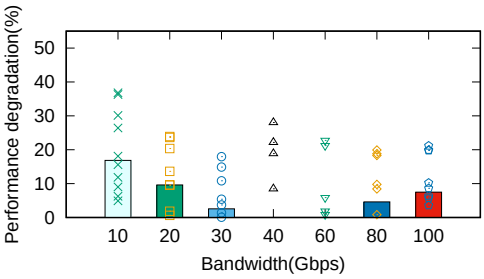


(d) ResNet152 の帯域幅による性能低下率

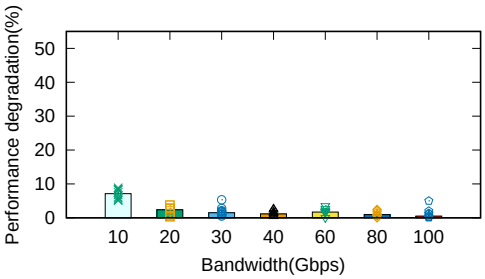
図 8: 最大 30% 以上の性能低下がみられるグループの評価対象ごとの帯域幅による性能低下率



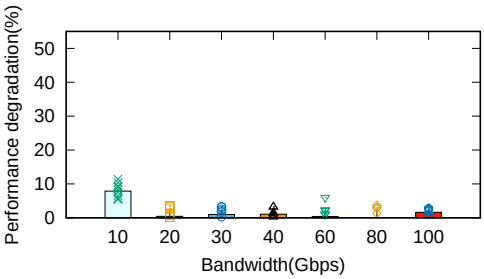
(a) NASNetLarge(Tensorflow Lite) の帯域幅による性能低下率



(b) ResNet50 の帯域幅による性能低下率

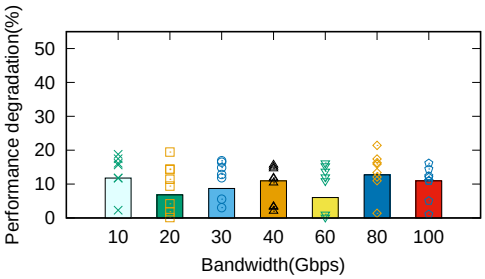


(c) ResNet101(Tensorflow Lite) の帯域幅による性能低下率

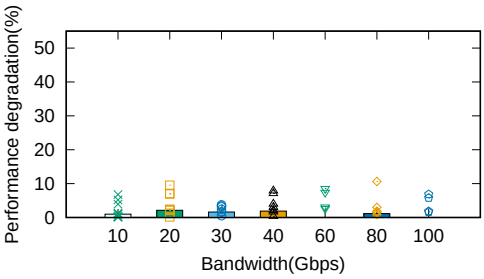


(d) ResNet152(Tensorflow Lite) の帯域幅による性能低下率

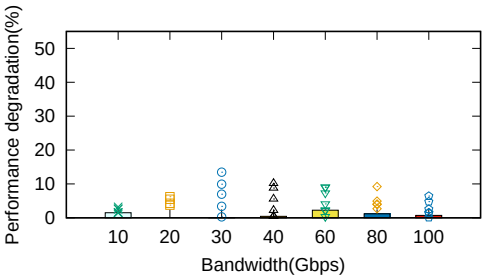
図 9: 最大 20% ほど性能低下がみられるグループの評価対象ごとの帯域幅による性能低下率



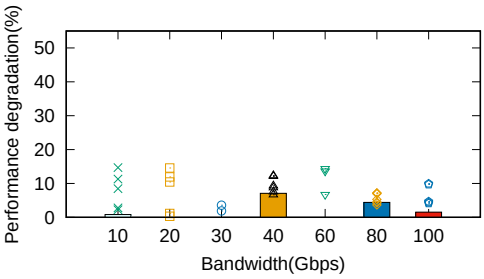
(a) ResNet50(Tensorflow Lite) の帯域幅による性能低下率



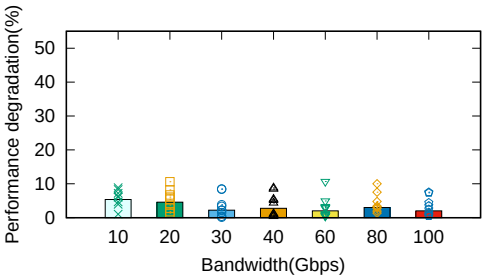
(b) MobileNetV2 の帯域幅による性能低下率



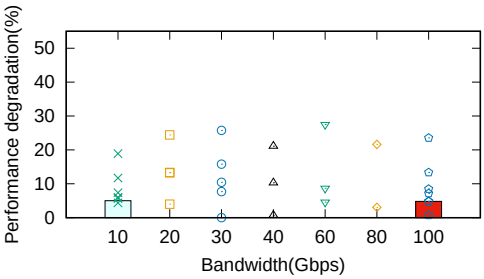
(c) MobileNetV2(Tensorflow Lite) の帯域幅による性能低下率



(d) NASNetMobile の帯域幅による性能低下率

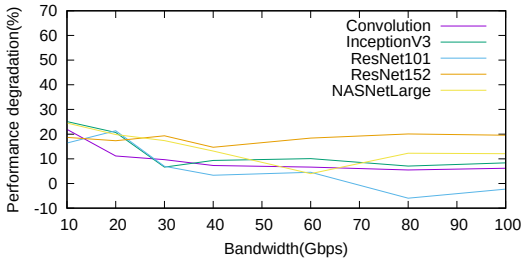


(e) NASNetMobile(Tensorflow Lite) の帯域幅による性能低下率

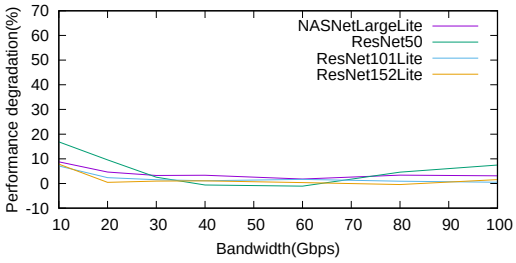


(f) ResNet50(Tensorflow Lite) の帯域幅による性能低下率

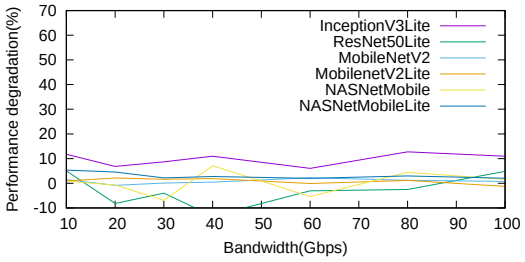
図 10: 遅延の影響を受けにくいグループの評価対象ごとの帯域幅による性能低下率



(a) 遅延の影響で 30% 以上の性能低下がみられるグループの帯域幅による性能低下率



(b) 遅延の影響で最大 20% ほど性能低下がみられるグループの帯域幅による性能低下率



(c) 遅延の影響をほとんど受けないグループの帯域幅による性能低下率

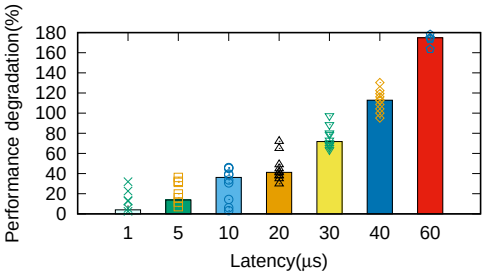
図 11: グループごとの帯域幅による性能低下率

図 11(c) から、遅延の影響をあまり受けなかったグループは帯域幅の値にによる影響をほとんど受けなかったことが分かる。そのため、これらのグループに関しては 10Gbps で十分であるといえる。次に、図 11(b) から、遅延の影響で最大 20% ほど性能低下がみられるグループは 40Gbps 以上の帯域幅では、性能低下の値が横ばいであったり、性能低下率の値が上下しているものもあるので、40Gbps 以上の帯域幅にしてもあまり影響はないといえる。最後に、図 11(a) から、遅延の影響で 30% 以上の性能がみられるグループは 40Gbps 以上の帯域幅では、性能低下の値が横ばいであるので、40Gbps で十分であるといえる。

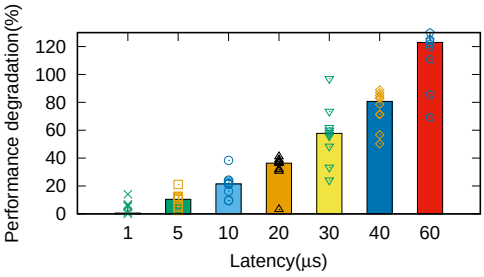
このことから、遅延の影響をほとんど受けないグループ以外は 40Gbps 以上の帯域幅が必要であるといえ、40Gbps 以上の帯域幅を用いたとしても、それで得られる性能向上の効果はわずかであるといえる。そのため、以降の調査では、遅延の影響をほとんど受けないグループを 10Gbps に固定し、それ以外のグループを 40Gbps に固定して調査を行う。

4.5.3　レイテンシによる性能低下率の評価

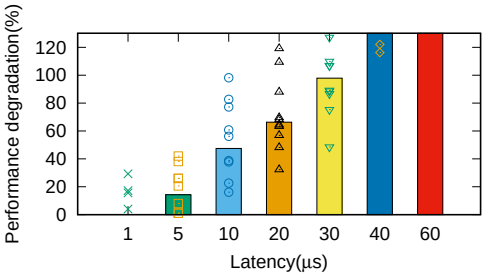
遅延の影響をほとんど受けないグループの帯域幅を 10Gbps に固定し、それ以外のグループの帯域幅を 40Gbps に固定してレイテンシの変更による性能低下率の調査した。このとき、レイテンシを、遅延の影響をほとんど受けないグループで 1μs、10μs、20μs、40μs、60μs、80μs、100μs に、それ以外のグループで 1μs、5μs、10μs、20μs、30μs、40μs、60μs に変更していき、そのときの性能低下率について調査した。結果を図 12 から図 14 に調査結果を示す。



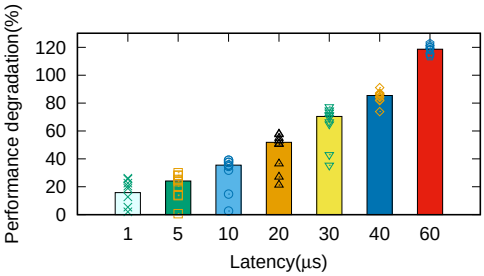
(a) NASNetLarge のレイテンシによる性能低下率



(b) InceptionV3 のレイテンシによる性能低下率

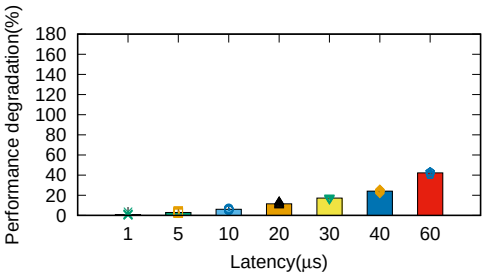


(c) ResNet101 のレイテンシによる性能低下率

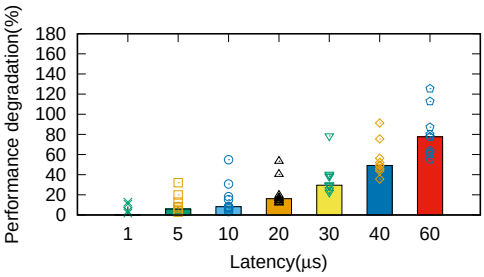


(d) ResNet152 のレイテンシによる性能低下率

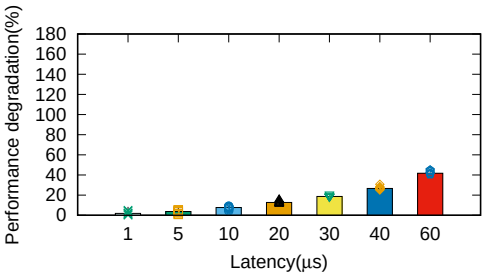
図 12: 最大 30% 以上の性能低下がみられるグループの評価対象ごとのレイテンシによる性能低下率



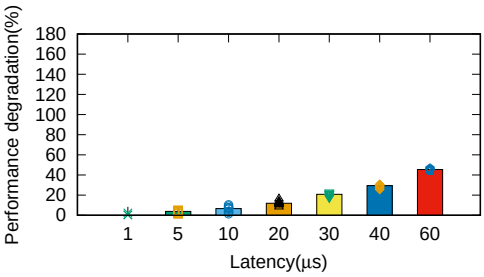
(a) NASNetLarge(Tensorflow Lite) のレイテンシによる性能低下率



(b) ResNet50 のレイテンシによる性能低下率

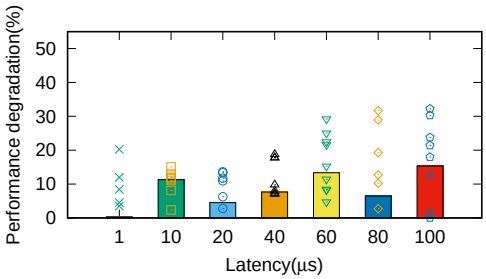


(c) ResNet101(Tensorflow Lite) のレイテンシによる性能低下率

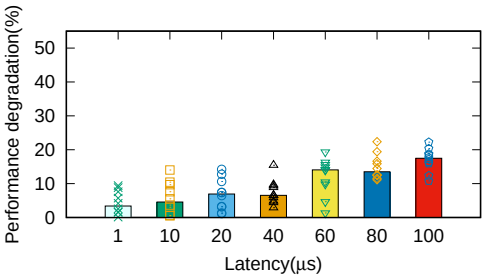


(d) ResNet152(Tensorflow Lite) のレイテンシによる性能低下率

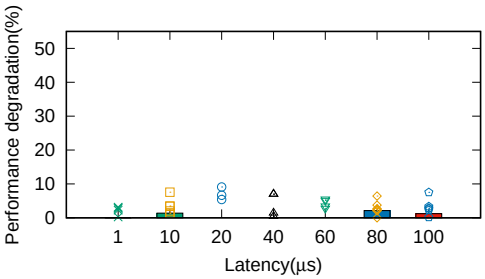
図 13: 最大 20% ほど性能低下がみられるグループの評価対象ごとのレイテンシによる性能低下率



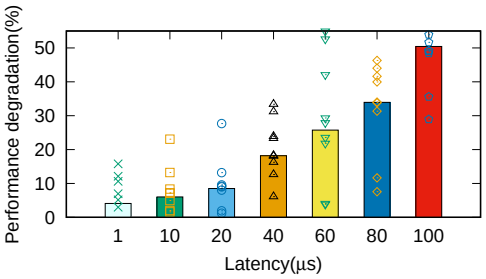
(a) ResNet50(Tensorflow Lite) のレイテンシによる性能低下率



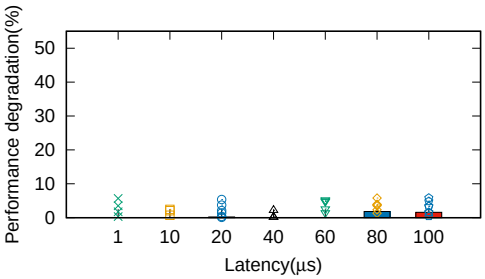
(b) MobileNetV2 のレイテンシによる性能低下率



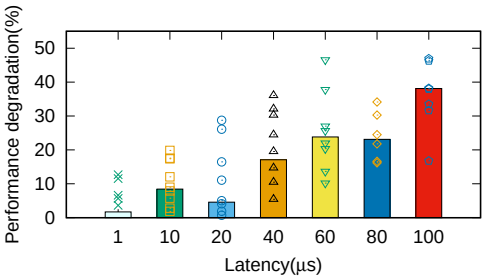
(c) MobileNetV2(Tensorflow Lite) のレイテンシによる性能低下率



(d) NASNetMobile のレイテンシによる性能低下率



(e) NASNetMobile(Tensorflow Lite) のレイテンシによる性能低下率

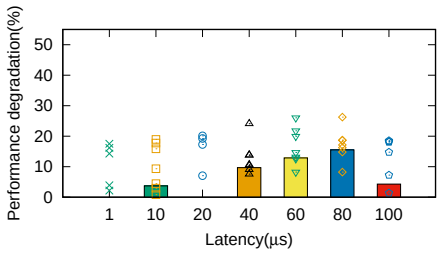


(f) ResNet50(Tensorflow Lite) のレイテンシによる性能低下率

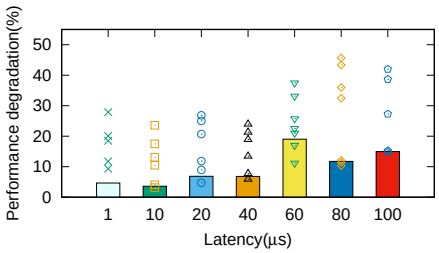
図 14: 影響を受けにくいグループの評価対象ごとのレイテンシによる性能低下率

この実験では、遅延の影響を受けにくいグループだけ 10Gbps に固定している。しかし、

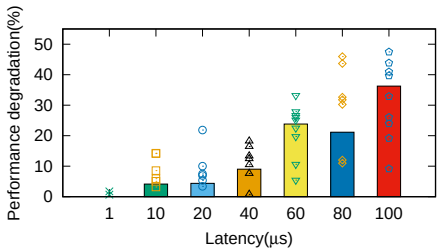
このグループにおいて、図 14(c) と図 14(e) に示している Tensorflow Lite の MobileNetV2 と NASNetMobile の二つのモデルとそれ以外のモデルでレイテンシによる性能低下の度合いが異なっている。具体的には、図 14(c) と図 14(e) は性能低下率がどのレイテンシでも低く、ほとんど影響を受けていないのに対し、図 14(a)、図 14(b)、図 14(d)、図 14(f) に示すグラフでは、レイテンシの影響を受けているようにみえる。そこで、Tensorflow Lite の ResNet50、InceptionV3、Tensorflow の NASNetMobile、MobileNetV2 の帯域幅を 40Gbps にして再び実験をした。その結果を図 15 に示す。



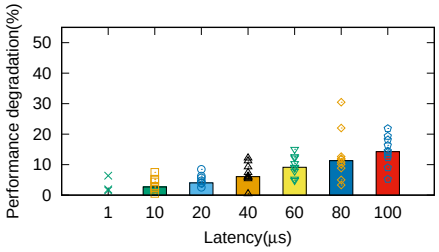
(a) ResNet50 のレイテンシによる性能低下率



(b) ResNet101 のレイテンシによる性能低下率



(c) ResNet152 のレイテンシによる性能低下率



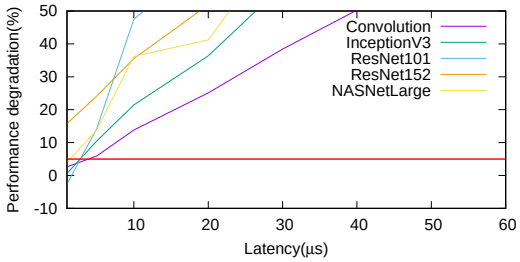
(d) InceptionV3 のレイテンシによる性能低下率

図 15: 帯域幅を 40Gbps にしたときの評価対象ごとのレイテンシによる性能低下率

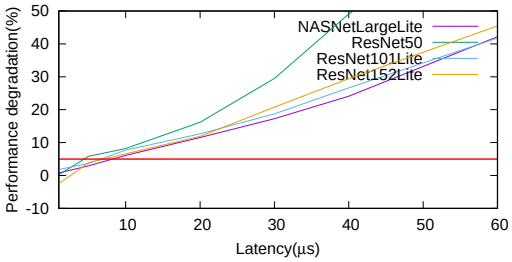
図 15 から、10Gbps の時よりも 40Gbps の時の方が性能低下を抑えられていることがわかる。図 15 に示したモデルは、帯域幅による違いがあまり得られず 10Gbps に固定していたが、レイテンシの影響による性能低下に対応するために、多少は性能低下を抑えられる 40Gbps の方が好ましいといえる。

ここで、分類したグループごとにそれぞれの性能低下率をグラフにしたものを図 16 に示

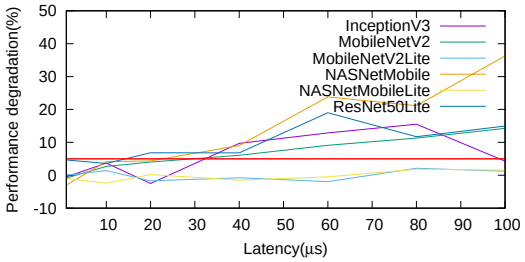
す。また、グラフ内の赤線は 5% を表している。



(a) 遅延の影響で 30% 以上の性能低下がみられるグループのレイテンシによる性能低下率



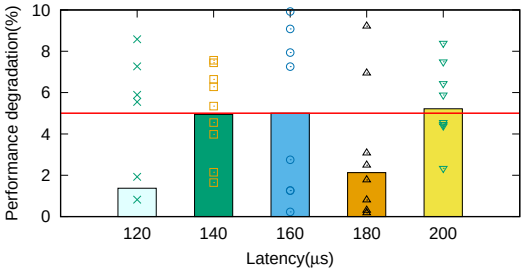
(b) 遅延の影響で最大 20% ほど性能低下がみられるグループのレイテンシによる性能低下率



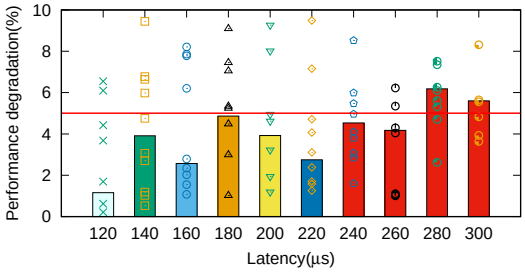
(c) 遅延の影響をほとんど受けないグループのレイテンシによる性能低下率

図 16: グループごとのレイテンシによる性能低下率

図 16(a) から、Tencsorflow Lite の MobileNetV2 と NASNetMobile はレイテンシが 100μs あっても 5% 以内に抑えられていることが分かる。そこで、この二つのモデルに対してさらにレイテンシを大きくして性能低下率合いを計測した。その結果を図 17 に示す。



(a) MobileNetV2(Tensorflow Lite) のレイテンシによる性能低下率



(b) NASNetMobile(Tensorflow Lite) のレイテンシによる性能低下率

図 17: レイテンシを 100μs 以上にした時の性能低下率

図 17 の赤色の横線は 5% を表している。Tencsorflow Lite の MobileNetV2 と NASNet-Mobile において、5% を超えた時点でのレイテンシはそれぞれ 160μs と 280μs である。このことから、レイテンシ要件をそれぞれ 140μs と 260μs とする。

ここで、性能低下を 5% 以内に抑えられるような帯域幅とレイテンシの評価対象ごとの最低の要件を表 3 に示す。

表 3: 評価対象の帯域幅とレイテンシ要件

評価対象	帯域幅	レイテンシ
NASNetMobile（Tensorflow Lite）	10Gbps	260μs
MobileNetV2（Tensorflow Lite）	10Gbps	140μs
InceptionV3（Tensorflow Lite）	40Gbps	20μs
ResNet50（Tensorflow Lite）	40Gbps	10μs
MobileNetV2	40Gbps	20μs
NASNetMobile	40Gbps	20μs
ResNet101（Tensorflow Lite）	40Gbps	5μs
ResNet152（Tensorflow Lite）	40Gbps	5μs
NASNetLarge（Tensorflow Lite）	40Gbps	5μs
ResNet50	40Gbps	1μs
NASNetLarge	40Gbps	1μs
InceptionV3	40Gbps	1μs
ResNet101	40Gbps	1μs
ResNet152	なし	なし

4.6 考察

今までの評価結果をもとに、性能低下率と評価対象との関係、リモートメモリの配置について、 リモートメモリの割当 についての考察を行う。

4.6.1 機械学習モデルと性能低下率の関係

図 6 の結果によって、メモリ分解の影響を大きく受けるのは Tensorflow で実行したときの ResNet50、ResNet101、ResNet152、NASNetLarge、InceptionV3 と Tensorflow Lite で実行したときの ResNet101、ResNet152、NASNetLarge であることが分かった。ここで、図 18 に、Tensorflow と Tensorflow Lite における性能低下率とモデルサイズの比較のためのグラフを示す。また、TensorFlow と TensorFlow Lite で動作させるモデルとサイズは表 1 に、各モデルとそのパフォーマンス低下率を表 5 に示している。性能低下率は、図 6 に示した結果において一番メモリ分解の影響を強く受けるレイテンシが 20μs、帯域幅が 10Gbps のときのものを採用した。

図 18 から、メモリ分解の影響をあまり受けていなかったモデルはモデルサイズが比較的小さいということが分かった。ただし、Tensorflow と Tensorflow Lite で影響を受け始めるモデルサイズは異なり、Tensorflow Lite の場合は 91MB の Inception-v3 まで影響を受けていないのに対し、Tensorflow で影響を受けていないのは 23MB の NASNetMobile 以下のモデルだけである。これは Tensorflow Lite は内部変数を小さくして計算するので、より大きなモデルでもメモリをあまり使用せずに実行できるからだと考えられる。ただし、モデル全体で見たとき、Tensorflow Lite でのモデルのサイズとパフォーマンスの低下率の相関係数は 0.861147 と正の相関があるといえるが、Tensorflow でのモデルのサイズとパフォーマンスの低下率の相関係数は 0.266956 であり、相関があるとは言えず、他の原因があると考えられる。

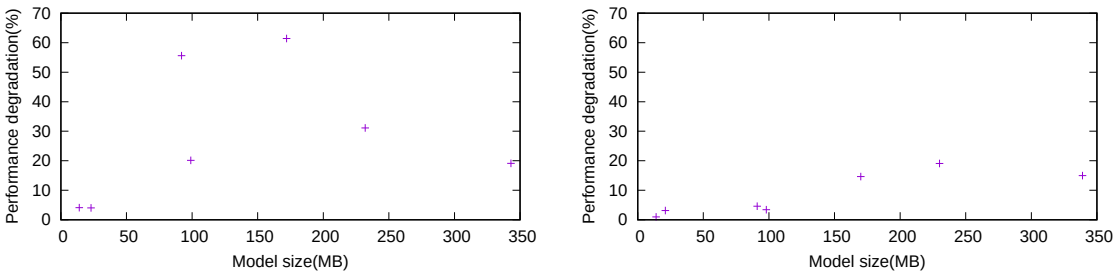
ここで、4.4 章から畳み込み計算による処理は行列のサイズがある程度大きい場合、リモートメモリの使用による遅延の影響を強く受けていることが分かる。また、本報告で評価対象とした機械学習モデルはすべて畳み込みニューラルネットワークを使用したモデルである。そこで、Tensorflow と Tensorflow Lite における畳み込み計算を行う層の総数と性能低下率との比較を図 19 に示す。また、本報告で使用するモデルでは畳み込み層として Conv2D、DepthwiseConv2D、SeparableConv2D の 3 種類の畳み込み層がある。ここで、各モデルとその畳み込み層の数を表 5 に示す。DepthwiseConv2D、SeparableConv2D は一部のモデルでしか使用されていないので、比較には Conv2D の数を用いる。

このとき、Tensorflow においては相関係数が 0.655662 であり、モデルのサイズと比べて正の相関がみられ、畳み込み計算の処理とパフォーマンスの低下には関係があるといえる。ここで、4.4 章から、畳み込み計算はある程度大きいサイズの行列の場合はメモリ分解の影響を大きく受けていることが分かる。また、表 5 に示している総パラメータ数を見ると、リモートメモリの影響を受けずらい NASNetMobile と MobileNetV2 を除いて、InceptionV3 の 23851784 が最小である。これは、4.4 章で示した、リモートメモリの影響を受けにくいサイズである 2048 × 2048 の行列のパラメータ数 4194304 よりもはるかに大きい数字であり、畳み込み計算の影響を受けやすいといえる。そのため、パフォーマンス低下を抑えるためには、畳み込み計算の計算量を減らすことが重要となるといえる。ただし、NASNetLarge に関しては SeparableConv2D の数と Conv2D の数を合わせると一番畳み込み処理をしているモデルであるといえるが、パフォーマンスの低下率がそれほど高くない。これは、SeparableConv2D はパラメータ数が少ない 2 つの畳み込み計算を重ねるという処理をしているため、計算の負荷が小さく、影響がでにくかったのではないかと考えられる。

このことから、畳み込みニューラルネットワークにおいては、畳み込み処理の量はメモリの使用量に影響を及ぼしていることが分かる。また、畳み込み層の多いモデルはリモートメモリによる遅延の影響を受けやすいといえる。

表 4: 各モデルとその性能低下率

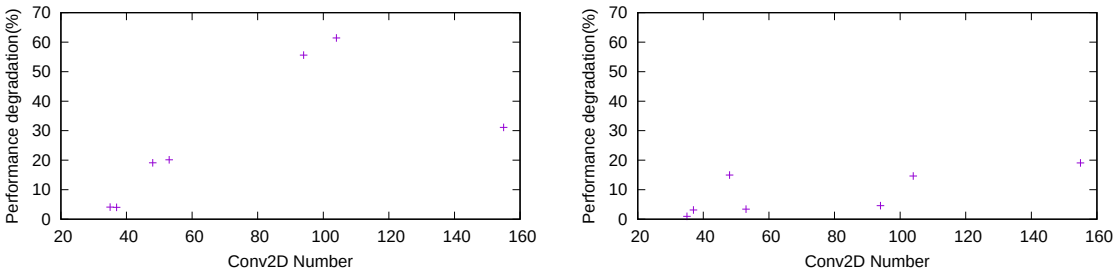
モデル	TensorFlow での性能低下率 (%)	TensorFlowLite での性能低下率 (%)
ResNet50	20.11849779	3.414003092
ResNet101	61.42124685	14.65268705
ResNet152	31.11816723	19.08513547
NASNetLarge	19.11323493	14.96181357
NASNetMobile	4.032566375	3.145361976
MobileNetV2	4.107053421	0.988419851
InceptionV3	55.59053732	4.601788618



(a) Tensorflow における性能低下率とモデルサイズの比較

(b) Tensorflow Lite における性能低下率とモデルサイズの比較

図 18: 性能低下率とモデルサイズの比較



(a) Tensorflow における性能低下率と Conv2D の数の比較

(b) Tensorflow Lite における性能低下率と Conv2D の数の比較

図 19: 性能低下率と Conv2D の数の比較

表 5: 各モデルと対応する畳み込み層の数と総パラメータ数

モデル	Conv2D の数	SeparableConv2D	DepthwiseConv2D	総パラメータ数
ResNet50	53	0	0	25636712
ResNet101	104	0	0	44707176
ResNet152	155	0	0	60419944
NASNetLarge	48	220	0	88949818
NASNetMobile	37	159	0	5326716
MobileNetV2	35	0	17	3538984
InceptionV3	94	0	0	23851784

4.6.2 リモートメモリの配置

本節では、リモートメモリの配置を考えるにあたり、簡単のため、CPU・メモリ間で発生する遅延は伝搬遅延のみとして考え、どれだけ離れた位置にあるメモリを利用できるのかについて考察を行う。伝搬遅延は 200m あたり 1μs である。表 3 から、必要とされるレイテンシ要件は 260μs、140μs、20μs、10μs、5μs、1μs となっている。このとき、レイテンシは一方のリソースへデータを送信しその返答が返ってくるまでの時間である。データの処理時間を 1μs ほどに仮定すると、データが往復するのに必要な時間はそれぞれ 259μs、139μs、

19μs、9μs、4μs、0μs となる。これらの値と伝搬遅延からリソースをどの範囲で配置できるのかを考察する。

1μs 以下のレイテンシ要件をもつ処理に関しては、データの往復時間が 0 に近い値でなければならないので、自身が持つローカルメモリやリモートメモリによって処理されるべきであり、他のリソースを使用することはできないといえる。レイテンシ要件が 5μs 以上の処理に関しては、生じる遅延が伝搬遅延のみであるならば、400m 以内の範囲にあるリモートメモリを活用しても性能低下は 5% 以内に抑えられる。同様にして、各レイテンシ要件から活用できるリモートメモリの設置範囲を導出できる。ここで、レイテンシ要件と距離範囲の対応を表 6 に示す。

表 6: レイテンシ要件と距離範囲の対応表

レイテンシ	距離範囲
1μs	0m 範囲内
5μs	400m 範囲内
10μs	900m 範囲内
20μs	1.9km 範囲内
140μs	14.9km 範囲内
260μs	26.9km 範囲内

表 3 からモバイル、エッジ端末向けの Tensorflow Lite やモデルサイズがとても小さいモデルでない限りは、5μs 以下のレイテンシが要求される。これは表 6 から 400m 以内のリモートメモリしか使用できない。そのため、少なくともリソースから 400m 以内にはリモートメモリを設置する必要があるといえる。

4.6.3 リモートメモリの割当

リモートメモリとのやり取りが多い処理ほど、リモートメモリを用いることによる性能低下が大きい。その反面、リモートメモリとのやり取りが多い処理は必要とするメモリサイズも大きい。そのため、そのような処理には、近くのリモートメモリや自身のメモリを優先的に割り当てる必要がある。一方で、リモートメモリとのやり取りが少ない処理は、遠くのリモートメモリを割り当てても性能低下が少ない。そのため、リモートメモリの割り当てのみならず、処理を実行する割り当ても含め、リモートメモリとのやり取りが多い処理については CPU・リモートメモリが近くになるように割当を行い、その残余資源を用いてそれ以外

の処理を行うといった割当が適当であると考えられる。このような割り当てを行うための具体的な割り当てアルゴリズムは、今後の課題である。

5　おわりに

本報告では、リモートメモリを用いたリソース分離型マイクロデータセンターを想定し、リモートメモリはリソース分離型マイクロデータセンターの性能にどれほど影響があるのかについて評価し、その結果を報告した。リモートメモリとの通信に必要とされる帯域幅は、40Gbps 以上必要であることが示された。また、レイテンシの要件は 1μs から 260μs まであり、実行する処理ごとに様々であるということが示された。

本報告では、行列計算と機械学習モデルによる処理を用いて評価を行った。しかし、本報告で用いた評価対象はリソースの分離について考慮されていなく、リソース分離されていない計算機での処理を想定したプログラムであるといえる。具体的には、キャッシュヒット率が高くなるようにプログラムの命令のアクセスパターンを整理することや、計算時に値の記憶を行わないようにして、メモリの使用量を減らすことが考えられる。そのため、報告したネットワーク要件はさらに緩くなる可能性がある。また、今後の課題として、一度にやり取りするデータのサイズがあげられる。本報告ではページサイズを 4KB に設定したが、ページサイズを大きくすることで空間局所性の観点からリモートメモリとやり取りする回数は少なくなり、レイテンシの影響が小さくなると考えられる。ただし、同時に一度にやり取りするデータサイズは増えるため、帯域幅による遅延の影響を受けやすくなる。また、データ転送の際にデータを圧縮することで、帯域幅による遅延の影響を小さくすることが可能である。ただし、データを展開する必要があるため、データの展開時間と帯域幅の遅延による影響との兼ね合いを考える必要がある。このように、リモートメモリとやり取りするデータのサイズについての評価は今後の課題としてあげられる。

謝辞

本報告を終えるにあたり、お忙しい中熱心にご指導、ご教授いただきました大阪大学大学院情報科学研究科の村田正幸教授に心より深く感謝を申し上げます。また、本研究を進めるにあたって、研究の方針や内容を含め様々な場面で適切な助言及びご指導をいただきました大阪大学大学院情報科学研究科の大下裕一准教授に心より感謝申し上げます。さらに、平素から広くご指導いただきました大阪大学大学院情報科学研究科の荒川伸一准教授、大阪大学大学院経済学研究科の小南大智助教、大阪大学大学院情報科学研究科の大歳達也特任助教に厚く御礼申し上げます。また、本報告に類似した事例として CPU と GPU との通信について、データのやり取りをする際の工夫点についてご教授いただきました大阪大学大学院情報科学研究科の置田真生准教授に心より感謝申し上げます。最後に、日頃より様々な面で支えていただきました山内雅明氏、松田拓己氏、安世民氏をはじめ、村田研究室の皆様に感謝の意を表して謝辞といたします。

参考文献

[1] 田中 裕之, 高橋 紀之, 川村 龍太郎, "IoT 時代を拓くエッジコンピューティングの研究開発," *NTT技術ジャーナル*, pp. 59–63, Aug. 2015.

[2] K. Bilal, O. Khalid, A. Erbad, and S. U. Khan, "Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers," *Computer Networks*, pp. 94–120, Jan. 2018.

[3] S. Han, N. Egi, A. Panda, S. Ratnasamy, G. Shi, and S. Shenker, "Network support for resource disaggregation in next-generation datacenters," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, pp. 1–7, Nov. 2013.

[4] Y. Cheng, R. Lin, M. De Andrade, L. Wosinska, and J. Chen, "Disaggregated data centers: Challenges and tradeoffs," *IEEE Communications Magazine*, Mar. 2019.

[5] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, "Network requirements for resource disaggregation," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 249–264, Nov. 2016.

[6] G. Zervas, H. Yuan, A. Saljoghei, Q. Chen, and V. Mishra, "Optically disaggregated data centers with minimal remote memory latency: Technologies, architectures, and resource allocation [invited]," *IEEE/OSA Journal of Optical Communications and Networking*, pp. A270–A285, Feb. 2018.

[7] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with infiniswap," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pp. 649–667, Mar. 2017.

[8] M. Bielski, I. Syrigos, K. Katrinis, D. Syrivelis, A. Reale, D. Theodoropoulos, N. Alachiotis, D. Pnevmatikatos, E. H. Pap, G. Zervas, V. Mishra, A. Saljoghei, A. Rigo, J. F. Zazo, S. Lopez-Buedo, M. Torrents, F. Zyulkyarov, M. Enrico, and O. G. de Dios, "dredbox: Materializing a full-stack rack-scale system prototype of a next-generation disaggregated datacenter," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1093–1098, Mar. 2018.

[9] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang, "Legoos: A disseminated, distributed OS for hardware resource disaggregation," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 69–87, Oct. 2018.

[10] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, June 2018.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, June 2016.

[12] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8697–8710, June 2018.

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, June 2016.

[14] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016.

[15] "Deploy machine learning models on mobile and IoT devices." `https://www.tensorflow.org/lite`, last accessed on 2020-2-8.

[16] "Keras: Deep learning for humans." `https://github.com/keras-team/keras`, last accessed on 2020-2-8.