社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

マスタ・ワーカ型パラメータサーバの実装とBESOMへの適用

黎　明曦[†,††]　谷村　勇輔[††,†]　一杉　裕志[††]　中田　秀基[††,†]

† 筑波大学　〒 305–8577 茨城県つくば市天王台 1-1-1
†† 産業技術総合研究所　〒 305–8568 茨城県つくば市梅園 1-1-1
E-mail: †{rei-meigi,yusuke.tanimura,y-ichisugi,hide-nakada}@aist.go.jp

あらまし　大脳皮質の計算論的モデルとして BESOM を提案している。BESOM は最新の神経科学的な知見に基づいてベイジアンネットで大脳皮質をモデル化したもので、人間の脳の比較的忠実なモデルとなっていることから、科学的に興味深いだけではなく、工学的にも広い応用が期待できる。その一方、BESOM における学習には、近年注目を集めているニューラルネットによるディープラーニングよりもさらに計算量が大きいことから、並列化による実行速度向上が要請される。本稿では、汎用のマスタ・ワーカ型タスク実行フレームワークを用いて、学習パラメータを交換するパラメータサーバを構築し、複数の BESOM モデルが協調して並列に学習する環境を実現した。これに、計算機内でのスレッド並列を組み合わせることで、16 台 4 スレッドを用いた場合で、最大 40 倍程度の高速化が可能であること、また、高速化の度合いは並列実行時のバッチサイズに依存することを確認した。
キーワード　BESOM, マスタ・ワーカ, パラメータサーバ

An Implementation of the Master-Worker based Parameter Server and its Application to BESOM

Mingxi LI[†,††], Yusuke TANIMURA[††,†], Yuuji ICHISUGI[††], and Hidemoto NAKADA[††,†]

† University of Tsukuba　Tennoudai 1–1–1, Tsukuba, Ibaraki, 305–8577 Japan
†† National Institute of Advanced Industrial Science and Technology　Umezono 1–1–1, Tsukuba, Ibaraki, 305–8568 Japan
E-mail: †{rei-meigi,yusuke.tanimura,y-ichisugi,hide-nakada}@aist.go.jp

Abstract　We have been proposing a computational model of the cerebral cortex called BESOM. BESOM models the cerebral cortex as Bayesian network based on recent findings in the neuroscience area. Because of its (relatively) faithful modeling, BESOM is not only scientifically interesting but also expected to be practically useful. On the other hand, machine learning with BESOM is a quite computation intensive task, even compared with the deep learning with conventional neural networks, that attract public attention recently. This means that parallel computation with multiple machines is essential for BESOM. In this paper we implement a parameter exchange mechanism, publically known as 'parameter server', using a general-purpose master-worker task execution mechanism, and constructed multi-node BESOM execution environment using it. We confirmed the it shows 40 times faster execution time with 16 machines and 8 threads each.
Key words　BESOM, Master-Worker, parameter server

1. はじめに

人間の脳を理解し、その情報処理原理を解明することは、人間のような知能を持つロボットを作るための一つの方法である。近年、神経科学の発展により、人間の脳への理解は著しく進んでおり、大脳皮質の情報処理とベイジアンネットの類似点が多数見つかっている。われわれはその知見に基づいて、計算機で大脳皮質の機能を再現することを目指している。BESOM モデル (BidirEctional Self Organizing Maps) [1] [2] [3] は、大脳皮質をベイジアンネットでモデル化した機械学習モデルである。BESOM は、大脳皮質に存在するマクロコラムを**ノード**として表現し、これらのノードを結んだ**ベイジアンネット**によって大脳皮質をモデル化するもので、パターン認識や強化学習など、人間の脳で行われる処理を実現することを目指している。

一般に機械学習は、大量の学習データを処理しなければならないことから、計算機に対する負荷が大きい。それに加えてBESOMは、ベイジアンネット上で高機能な推論動作を実現するために確率伝搬法と呼ばれるある種の反復計算を用いるため、一般的なニューラルネットと比較して、より負荷が大きい。したがって、BESOMを実用的に利用するには、並列化による高速化が不可欠である。

機械学習を並列化する手法には、大別して、ひとつの機械学習モジュールの内部を並列化する方法と、複数の機械学習モジュールを並列に実行する方法が考えられるが、本稿では後者を用いる。具体的には、複数の機械学習モジュールを独立、並列に動作させ、それらの間で定期的に学習パラメータの調停を行うことで、並列に学習を進行させる。このパラメータの調停には、一般にパラメータサーバと呼ばれる機構を用いるが、これを汎用のマスタワーカ型のタスク処理フレームワークで実現する。さらに、単一計算機内部でも複数のスレッドを用いて複数の機械学習モジュールを並列実行する。

評価には、機械学習の標準的なワークロードの一つであるMNIST手書き文字認識を用い、利用台数、スレッドあたりの速度向上率を計測する。また、調停を行う実行間隔に相当するバッチサイズと、速度向上率の相関を調査する。

本論文の構成は以下の通りである。2. でBESOMを概説する。3. ではパラメータサーバの実装について述べる。4. で評価を行う。5. で関連する研究を議論する。6. では、まとめと今後の課題について述べる。

2. BESOMとその並列化

2.1 BESOMの概要

計算論的神経科学における注目すべき進展として、「大脳皮質がベイジアンネットである」という仮説の登場がある。大脳皮質とベイジアンネットは、機能と構造の両面で多くの類似性を持ち、実際にさまざまな神経科学的現象がベイジアンネットを用いたモデルで再現されている。このような観点から、われわれは大脳皮質をベイジアンネットでモデル化した計算論的モデルBESOMを提案している。

ベイジアンネットを用いたディープラーニングは以下の点から有望であると思われる。

- ベイジアンネットは複数の事象の間の因果関係を確率により表現する知識表現の技術である。ベイジアンネットは信号源と観測データの間の因果関係を比較的少ないメモリで完結に表現できる場合があり、その場合は少ない計算量でさまざまな推論を行うことができる。
- 推論動作は、ネットワーク全体の情報を使って行われる。入力からのボトムアップの情報だけでなく、文脈からのトップダウンの予想の情報も用いたロバストな認識を行うことができる。したがって、フィードフォワード型ニューラルネットよりもはるかに高機能である。
- 階層的な生成モデルを素直に表現できるため、学習対象の事前知識が作りこみやすい。事前知識をネットワーク構造やパラメタ事前分布の形で作りこむことで、学習の性能を上げられる可能性がある。

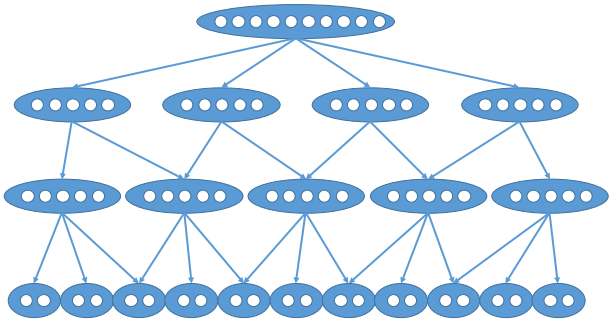

図1　BESOMのネットワーク例

BESOMは、大脳皮質に存在するマクロコラムをノードとして表現し、これらのノードを結んだベイジアンネットによって大脳皮質をモデル化する。個々のノードは複数の**ユニット**から構成される。ノードは確率変数に相当し、ユニットはその確率変数が取りうる値を示す。

図1にBESOMのネットワークの例を示す。楕円がノード、その中の白丸がユニットを表現している。ネットワークはこの図で示すように、一般に多階層の構造をとる。

ソフトウェアとしてのBESOMは、Javaで記述されており、パラメータをインタラクティブに操作するためのGUIを備えている。

2.2 BESOMの並列化

BESOMの学習の並列化に関しては、大別して2つの方針が考えられる。

ひとつは単一のモデル内での学習を並列化する方法である。BESOMでの学習には、その一部に確率伝搬法を用いるが、この計算は個々のノードに対して独立に行う事ができるため、ノード数分の並列度が存在し、比較的容易に並列化を行うことができる。

もう一つの方法は、複数のモデルを用いて独立、並列に学習を進める方法である。学習中に定期的にモデル上の学習パラメータを収集し、調停した結果を再配布してモデル間の同期を取ることで、効率的に学習をすすめることができる。

もちろんこの2つは排他的ではなく、双方を同時に実現することも可能である。

本稿では、後者の方法を用いて並列化を行った。これは、前者の方法が複数計算機間に拡張することが比較的難しいのに対し、後者の方法は、単一計算機内でのマルチスレッド、マルチプロセスによる並列化にも複数計算機を用いた分散実行にも容易に拡張可能だからである。

複数モデルを用いる場合の重要なパラメータとしてバッチサイズがある。これは、ひとつのモデルが他のモデルと通信せずに学習を行う回数をあらわす。バッチサイズが大きすぎると、個々のモデルが異なるローカルミニマムに収束してしまい、調停した結果が無意味なものになり、学習の進度が低下する可能性がある。一方で、バッチサイズが小さいと、頻繁にモデル間の通信が発生する事になるため、実行速度が低下することが予測される。

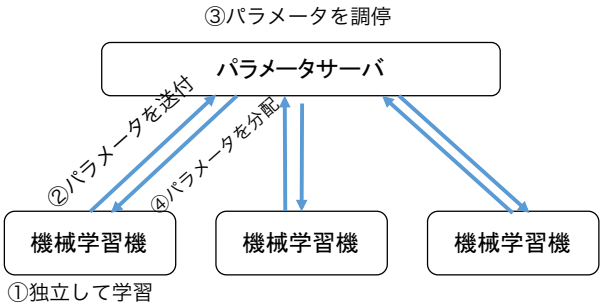

図 2　パラメータサーバの動作

```
public interface TaskManager {
  // 初期化。
  void initialize(String initialString);

  // タスクを生成。タスクがなければブロックする。
  // 終了時には null を返す。
  List<Task> getTasks();

  // タスクの結果を受け取る。
  void putResult(TaskResult tr);
}


public interface Solver {
  // 初期化。
  void initialize(String initialString);

  // タスクの処理。
  TaskResult solve(Task t);
}
```

図 3　TaskManager と Solver のインターフェイス

3.　パラメータサーバの実装

機械学習を並列に進める際に、パラメータを共有する機構として、パラメータサーバと呼ばれる機構が提案されている [4]。パラメータサーバは定期的に、独立して稼働する機械学習モジュールから定期的にパラメータを収集し、それらのパラメータ間の調停を行い、機械学習モジュールに分配する。この様子を図 2 に示す。

3.1　マスタ・ワーカ型のタスク管理システム

本稿では、パラメータサーバをマスタ・ワーカ型のタスク管理システムを用いて実装した。用いたマスタ・ワーカ型タスク管理システムは Java で記述されたシンプルなもので、タスクを生成する TaskManager とタスクを処理する Solver を生成するファクトリクラスを指定することで、さまざまな問題に対応することができる。TaskManager と Solver のインターフェイスを図 3 に示す。TaskManager はマスタ上で動作し、Solver はワーカ上で動作する。

3.2　マスタ・ワーカ型のタスク管理システムを用いたパラメータサーバの実装

パラメータサーバをマスタ・ワーカ型で実装するには、個々のワーカに機械学習モジュールを実装し、マスタでパラメータの調停を行えば良い。

パラメータの配布はマスタからワーカへ渡されるタスクの一部として行う。個々のワーカ（機械学習モジュール）でアップデートされたパラメータはタスクの結果として、マスタに回収する。一つのタスクで実行する学習回数をバッチサイズと呼び、

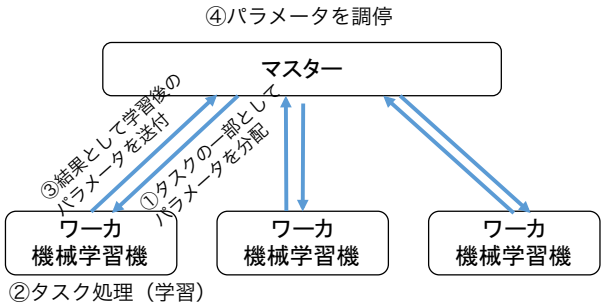

図 4　マスタ・ワーカで実装したパラメータサーバの動作

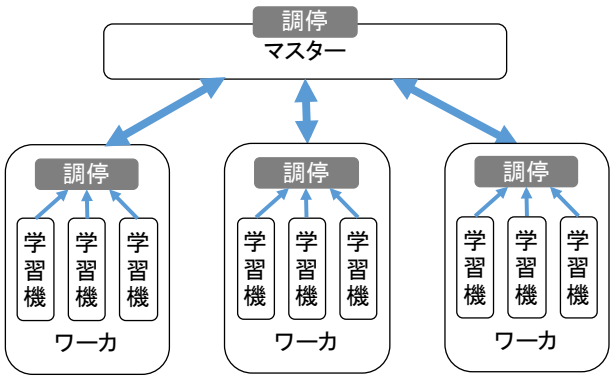

図 5　スレッド並列とマスタ・ワーカ並列の併用

タスクを表す情報の一部として与える。この様子を図 4 に示す。

動作をまとめると以下のようになる。

- マスタは、ワーカの数だけタスクを生成し、ワーカに与える (図 4(1))。このときタスクとしてパラメータを渡すが、は前回の実行で得られた結果に対して調停を行った結果のパラメータである。
- ワーカは、タスクとしてバッチサイズとパラメータを受け取る。パラメータを機械学習モジュールにセットし、バッチサイズの回数だけ学習操作を行う (図 4(2))。
- その結果更新された学習結果のパラメータを、タスクの結果としてマスタに返却する (図 4(3))。
- マスタは、結果として受け取ったパラメータが集まると、パラメータの調停を行う (図 4(4))。

この過程を、規定の回数繰り返す。

3.3　スレッド並列との併用

本稿では、マスタ・ワーカでの並列化に加えて、計算機内でのスレッド並列も併用する。並列化手法はマスタ・ワーカでの並列化と同様で、個々のスレッドが独立に学習を行い、定期的にパラメータの調停を行う手法を用いた。

スレッド間のパラメータ調停は、マスタでの調停に先立ってローカルに行われる。スレッド内の各学習機でバッチサイズ分の学習が終了すると、ワーカ計算機内で、各スレッドでの結果パラメータを集計し、調停を行う。その結果をワーカ計算機の結果として、マスタに返却する。つまり調停は、計算機内と計算機間とで階層的に行われることになる。この様子を図 5 に示す。

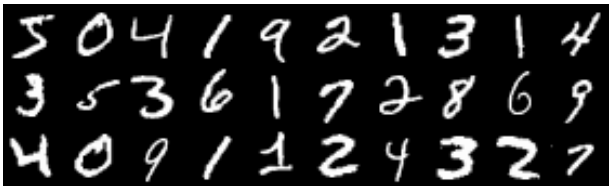

図 6　MNIST 手書き文字の例

表 1　バッチサイズと速度向上率

バッチサイズ	25	50	100
速度向上率	21.7	28.3	40.6

4.　評　　価

4.1　評価環境と評価項目

本研究での評価には、1 基のヘッド計算機と 16 基のワーカ計算機から構成される小規模なクラスタを使用した。各計算機は 1Gbit Ethernet で接続されている。CPU は Intel(R) Xeon(R) CPU W5590 （3.33GHz, 4 コア） x 2 ソケット、メモリは 48GByte を搭載している。カーネルは Linux 3.13.0 である。Java 処理系には Oracle JDK 1.8.0_45 を用いた。

評価実験には MNIST 手書き文字認識 [5] を用いた。これは 0 から 9 までの手書きの数字を 28x28 の白黒 8bit の画像としたラベル付きデータで、60000 件の訓練データと 10000 件のテストデータから構成される。図 6 に画像の例を示す。この教師画像 1 枚を学習することを 1 回と数え、特定の回数を学習する時間を計測した。

使用するワーカー計算機の数と、個々のワーカー計算機上で稼働するワークスレッドの数を変化させ、定数回の学習を行う時間を比較した。また、バッチサイズの速度向上率に与える影響を見るために、バッチサイズも変化させた。この際、マスタ上でのパラメータの調停にかかる時間も計測した。

学習回数は、バッチサイズ 25 および 50 では、6400 回とし、バッチサイズ 100 では 12800 回とした。

4.2　評 価 結 果

実行時間に対する評価は、ワーカーの台数を 1,2,4,8,16、ワーカーのスレッド数を 1,2,4,8 とそれぞれ変化させて、学習全体にかかった時間と、パラメータの調停にかかった時間を測定した。実験は 3 回行い、その平均をとった。

4.2.1　並列化による速度向上率

バッチサイズを 25、50、100 とした時の実行時間をそれぞれ表 2, 表 3, 表 4 に示す。バッチサイズ 100 の場合のみ、学習回数が倍となっていることに注意されたい。同じ結果をグラフにまとめたものをそれぞれ図 7, 図 8, 図 9 に示す。X 軸は実行に用いた総スレッド数、Y 軸は 1 ワーカ 1 スレッドで行った場合に対する相対的な速度向上率を示している。X 軸、Y 軸ともに対数表記となっている。

基本的な傾向はいずれの場合も同じで、ワーカー数が増えると、速度が向上している。ただし、スレッド数が増えれば必ず速度が向上するわけではなく、同じスレッド数では、ワーカー数の多い方、すなわちワーカあたりのスレッド数が小さいほうが効率が高いとわかる。

表 2　学習所要時間（バッチサイズ 25）(s)

スレッド数＼ワーカ数	1	2	4	8	16
1	256.9	131.0	68.3	37.7	23.9
2	140.6	75.0	40.0	22.3	15.3
4	86.5	45.4	25.0	15.5	11.9
8	89.7	53.2	27.1	15.3	12.5

表 3　学習所要時間（バッチサイズ 50）(s)

スレッド数＼ワーカ数	1	2	4	8	16
1	254.2	129.8	66.9	36.4	21.6
2	137.1	71.6	38.3	21.7	13.3
4	80.3	42.8	23.9	14.0	9.2
8	86.4	45.6	25.8	14.6	9.0

表 4　学習所要時間（バッチサイズ 100）(s)

スレッド数＼ワーカ数	1	2	4	8	16
1	560.9	257.7	131.1	68.9	37.9
2	276.4	142.9	72.6	38.5	21.9
4	150.5	81.2	42.3	23.1	13.8
8	165.3	87.0	47.0	24.9	14.3

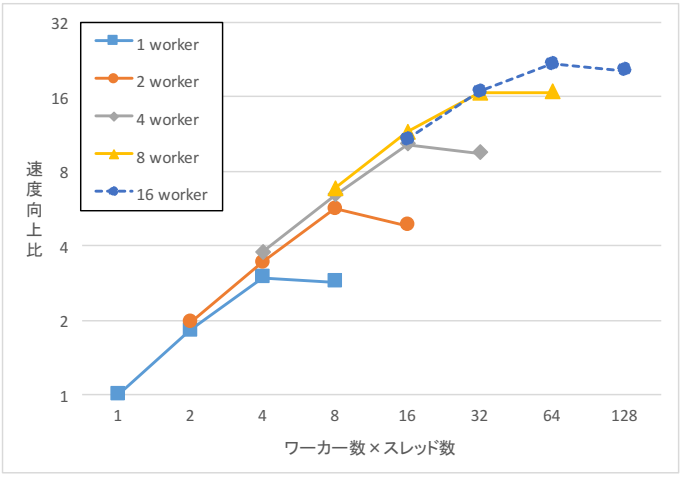

図 7　速度向上率（バッチサイズ 25）

また、多くの場合、同一ワーカ数に対して、8 スレッドを用いた場合より、4 スレッドを用いたほうが性能が高い。これは、8 スレッドの場合、メモリのバンド幅やキャッシュが不足し、逆に性能が低下したためであると考えられる。

バッチサイズと速度向上率の関係を見るために、各バッチサイズにおける最大速度向上率をまとめたものを表 1 に示す。バッチサイズを大きくすることで速度向上率が大きくなることがわかる。これは、パラメータ調停の回数が低減されるために、そのために必要となる通信と計算のコストが低減されるためである。

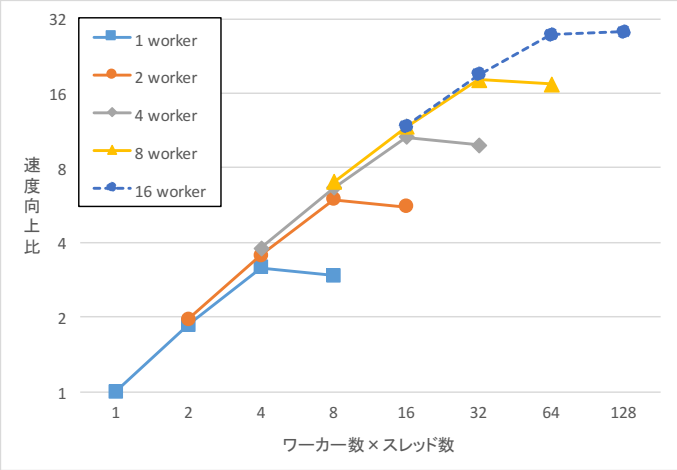

図 8　速度向上率（バッチサイズ 50）

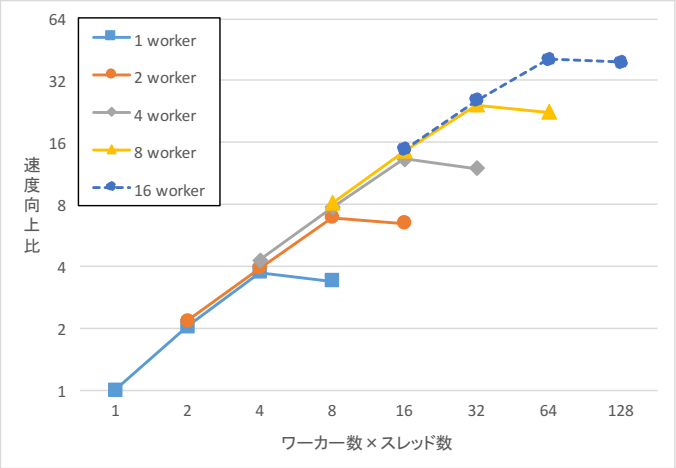

図 9　速度向上率（バッチサイズ 100）

4.2.2　ワーカ数とパラメータ調停にかかる時間

ワーカ数とパラメータ調停にかかる時間の相関を調べるために、1 ワーカあたり 1 スレッドの場合に、マスタ上で行われる調停にかかった時間とワーカ数を、表 5 にまとめた。調停の回数はワーカ数、スレッド数に依存するので、これを調停の回数で割り、一度の調停あたりの時間を算出したものを表 6 に示す。

ここで示す調停時間は、マスタ上で行われるもので、ワーカ上でローカルに行われる調停時間や、調停に必要となるデータ通信時間は含まれていないことに注意する必要がある。

表 6 を見ると、調停にかかる時間はワーカ数の増大にしたがって、ほぼ線形に増大している事がわかる。これは、現在の調停の実装が、パラメータの逐次的な加算による平均で実装されているためである。ワーカ数が十分小さいため、現状では特に問題にはならないが、ワーカ数が数百のオーダとなる場合には何らかの対策が必要である。対策としては、加算をツリー状に並列化することで、ワーカ数に対して対数オーダに抑える方法が考えられる。

5.　関 連 研 究

本稿ではパラメータサーバを一般名詞として用いているが、固有名詞としての Parameter Server も存在する [4]。この Parameter Server は、CMU のチームが開発したもので、大規模なパラメータ集合を取り扱うために、サーバが複数の計算機に分散している点が、われわれのシンプルなパラメータサーバとの最大の相違点である。個々のパラメータサーバは、パラメータの部分集合を担当する。このように分散する事で一つの計算機に対する通信の集中を防ぐことができる。また、通信量を削減するために、パラメータそのものではなくその勾配のみを通信する、通信時間を隠ぺいするために同期の制約を緩和する、といったさまざまな工夫がなされている。

表 5　調停総時間 (ms)

バッチサイズ＼ワーカ数	1	2	4	8	16
25	637	649	565	534	450
50	331	346	310	287	241
100	339	346	310	278	237

表 6　1 回の調停時間 (ms)

バッチサイズ＼ワーカ数	1	2	4	8	16
25	2.49	5.07	8.82	16.7	28.1
50	2.58	5.41	9.69	17.9	30.1
100	2.65	5.41	9.69	17.4	29.6

Project Adam [6] は、マイクロソフトが機械学習用に提案したパラメータサーバである。基本的な構成は上述の Parameter Server と類似するが、耐故障性を実現するために、パラメータサーバコントローラを持つ点に特徴がある。コントローラは Paxos クラスタを構成する複数の計算機からなり、パラメータの各部分のサーバへの割り当てを司る。コントローラはサーバの死活監視を行い、パラメータの再割当てを行う。各パラメータは 2 つ以上のサーバに複製されて保存される。このような構造は、大規模な分散ストレージでは一般的であるが、大規模な機械学習には数日以上の計算時間がかかるとはいえ、分散ストレージと比較すると、はるかに持続時間が短いパラメータサーバで、ここまでのコストをかけて耐故障性を実現する意味があるかは疑問である。

Yahoo の Smola らは高性能の分散式トピックモデル構築アーキテクチャを提案している [7] [8]。このアーキテクチャでは、分散して動作するスキャナの同期を、memcached を用いて実装した分散キーバリューストレージを用いた黒板モデルで行っている。各スキャナは非同期にそれぞれ分散キーバリューストレージに書き込み、グローバルな状態管理者が後処理として加算を行うモデルである。分散キーバリューストレージを用いることで、各計算機のネットワーク負荷が軽減される。また、データの書き込みと集計処理が分離されることで、同期コストが低減される。このアプローチが有効になるのは非常に規模が大きい場合のみであるが、われわれのパラメータサーバにも適用可能である。

6. おわりに

本稿では、BESOM モデルのマスタ・ワーカ型パラメータサーバを用いた並列化について述べた。並列化の結果、16 台 8 スレッドを用いておよそ 40 倍の高速化を実現することができた。また、バッチサイズによって、速度向上率が変化することを確認した。

今後の課題としては以下が挙げられる

- バッチサイズによる学習速度の相関

今回の議論では、学習の回数のみに着目し、実際の学習の進行は見ていないが、本来問題となるのは学習回数ではなく学習速度である。本稿ではバッチサイズと学習回数の相関を確認したが、実際にはバッチサイズと学習の速度には別の相関があり、これら双方を勘案した上で、最適なバッチサイズを決定する必要があると思われる。

- GPGPU を用いたモデル内並列化

2.2 で述べたとおり、今回の並列化では、スレッド並列化においても、複数のモデルを並列に学習する方法をとった。一方、モデル内部にも十分な並列性が存在することから、モデル内を並列化することが考えられる。ディープラーニングでは Caffe [9] に代表されるように、GPGPU を用いてモデル内を並列化することが広く行われているが、BESOM においても同様に GPGPU を用いることで、大幅な高速化が期待できる。

- モデルを複数計算機に拡張

今後 BESOM で扱うモデルのサイズが増大すると、計算量、メモリ容量の双方の観点で単一計算機内での計算が難しくなることが予想される。文献 [6] で行われているのと同様に、一つのモデルが複数計算機にまたがって実行できるように拡張することも、今後の課題の一つである。

文　献

[1] 一杉裕志, “大脳皮質のアルゴリズム besom ver.1.0,” AIST09 J00006, 産業技術総合研究所, 2009.

[2] 一杉裕志, “大脳皮質のアルゴリズム besom ver.2.0,” AIST11 J00009, 産業技術総合研究所, 2011.

[3] Y. Ichisugi and N. Takahashi, “An efficient recognition algorithm for restricted bayesian networks,” Proc. of 2015 International Joint Conference on Neural Networks (IJCNN 2015) (to appear)., pp.●●–●●, 2015.

[4] “Parameter server: http://parameterserver.org/”. Accessed: 2015-06-20.

[5] “The mnist database of handwritten digits http://yann.lecun.com/exdb/mnist/”. Accessed: 2015-06-20.

[6] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, “Project adam: Building an efficient and scalable deep learning training system,” 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pp.571–582, USENIX Association, Broomfield, CO, Oct. 2014. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/chilimbi

[7] A.J. Smola and S. Narayanamurthy, “An architecture for parallel topic models,” Very Large Databases (VLDB), pp.●●–●●, 2010.

[8] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A.J. Smola, “Scalable inference in latent variable models,” WSDM ’12: Proceedings of the fifth ACM international conference on Web search and data mining, pp.123–132, ACM, New York, NY, USA, 2012. http://dl.acm.org/authorize?6666391

[9] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R.B. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” CoRR, vol.abs/1408.5093, pp.●●–●●, 2014. http://arxiv.org/abs/1408.5093