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.

パラメータサーバを用いた並列機械学習システムにおける耐故障性のシミュレーション

黎　　明曦[†,††]　谷村　勇輔[††,†]　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　〒 305–8560 茨城県つくば市梅園 1-1-1
E-mail: †{rei-meigi,yusuke.tanimura,hide-nakada}@aist.go.jp

あらまし　大規模なデータを対象とする機械学習システムの高速化には並列化が必須である。パラメータサーバと多数のワーカ計算機を用いる**データ並列機械学習システム**においては、一般の大規模システムと同様に耐故障性が問題になるが、並列機械学習システムにおける耐故障性の議論は進んでいない。本稿ではパラメータサーバを用いた並列機械学習システムにおける耐故障性に関して議論し、シミュレーションを用いて大規模なシステムにおける定量的な評価を行う。その結果、パラメータサーバ上の情報を用いることでチェックポイントのコストを大幅に低減することができること、さらには、収束への悪影響を許容すれば、チェックポイントからのリカバリコストも低減できることを明らかにした。

キーワード　耐故障性、パラメータサーバ、機械学習、シミュレーション

A simulation study on fault tolerancy of parallel machine learning systems with parameter servers

Mingxi LI[†,††], Yusuke TANIMURA[††,†], 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,hide-nakada}@aist.go.jp

Abstract　Parallel computation is essential for machine learning systems to be more faster. There are two techniques to build parallel machine learning systems; namely data parallel method and model parallel method. In this paper, we only disuss data parallel where large number of parameter servers and computation servers communicate each other to perform computation. Fault tolerancy is a big problem on large scale computation system in general, however, there are not much discussions about the fault folerancy of parallel machine learning system. in this paper, we discuss the fault tolerancy of parallel machine learning systems which use parameter servers. Parameter servers gives extra redundancy to the system and could double as the checkpoint server. We also quantitatively evaluate several fault tolerance method using parallel environment simulator SimGrid.

Key words　fault tolerancy, parameter server, machine learning, simulation

1. はじめに

機械学習には大量の学習データを処理することが必要であるため、計算機への負荷が高く、並列化による高速化が必須である。機械学習システムを並列化する方法には大別して、複数の機械学習機がデータセットを分割して学習するデータ並列と呼ばれる手法と、一つの機械学習機の内部を並列化するモデル並列と呼ばれる手法の二種類がある。本研究では、パラメータサーバというパラメータ管理機構を用いたデータ並列機械学習システムを対象とする。

データ並列機械学習システムは、数百以上の計算機からなる大規模な並列システムである。一般にシステムの規模が増大すると、システムの故障可能性が増大するため、耐故障性を考慮したシステム構築が必要となる。一般的な並列システムの耐故

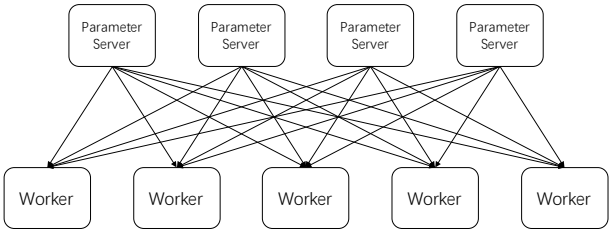

図 1　パラメータサーバを用いた並列機械学習機の構造

障性は、科学技術並列計算のコンテクストで広く研究されている。

パラメータサーバを用いた並列機械学習システムでは、類似した情報が多くのサーバに存在するため、一般的な並列システムの場合と比較して、障害復元時に必要な手順の多くを省くことができる。このような耐故障性は、個々のシステムでは個別に実装されているが、性能面での解析は十分ではない。

われわれは、パラメータサーバを用いた並列機械学習システム固有の対故障性手法について議論するとともに、分散並列環境シミュレータ SimGrid [1] を用いて、大規模環境を仮想的に構築し、定量的な評価を行った。通常の並列システムで行われる定期的なチェックポインティングと故障時のリカバリを組み合わせる方法と、パラメータサーバを用いた並列機械学習システム固有の対故障性手法 2 つを比較した。

本論文の構成は以下の通りである。2. に本研究の背景を示す。3. では本稿で用いた対故障性手法について述べる。4. に SimGrid を用いたシミュレーションとその結果を示す。5. はまとめである。

2. 背　　景

2.1　パラメータサーバを用いた並列機械学習システム

大規模な機械学習システムには、大規模な分散システムが必須である。例えば文献 [2] においては、16CPU コアを持つ計算機 1000 台を 3 日間もちいた学習を行っている。

機械学習システムを並列化によって高速化するには、大別して 2 つの手法がある。一つは、単一の機械学習機の内部を並列化する手法で、**モデル並列**と呼ばれる手法である。もう一つは、複数の機械学習機を、ゆるく同期した状態で並行して動作させることで高速化する手法で、**データ並列**と呼ばれる。個々の機械学習機は、データセットのそれぞれ異なるサブセットを学習し、パラメータを更新する。更新されたパラメータを定期的に集約して再分配することで、機械学習機群をゆるく同期させ、学習を加速する。

このような手法は文献 [2] でも用いられている [3]。パラメータを定期的に集約、再分配するために用いるサーバ機構を、一般にパラメータサーバ [4] [5] と呼ぶ。われわれは、これまでにマスターワーカを用いた簡易的なパラメータサーバを構築し、大脳皮質モデル BESOM [6] の高速化を行ってきた [7]。

図 1 にパラメータサーバを用いた並列機械学習機の構造を示す。システムは、パラメータサーバとワーカ（機械学習機）から構成される。ワーカは定期的にパラメータサーバにパラメータ更新情報を送信し、最新のパラメータを受け取る。パラメータサーバはワーカからの更新情報を受信し、内部に保持したパラメータを更新し、ワーカに返信する (図 2)。

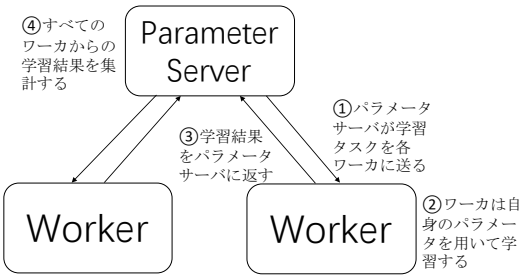

図 2　パラメータサーバの動作

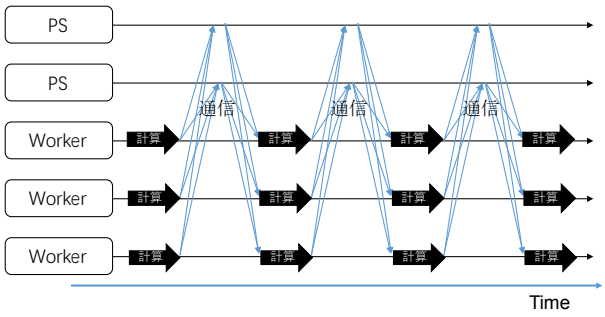

図 3　パラメータサーバの通信

図 3 に、パラメータサーバ群とワーカ群間の通信を時間軸に沿って示す。パラメータサーバとワーカの間では全対全で通信が行われる。計算全体としては一種の BSP（Bulk Synchronous Parallel [8]）を構成しており、ワーカ群がパラメータサーバ群を介して周期的に同期を繰り返す構造となっている。

パラメータサーバが複数あるのは、パラメータサーバがボトルネックになることを避けるためである。大規模な機械学習においてはパラメータの数が膨大になるため、単一のパラメータサーバではネットワーク通信ボトルネックになることが避けられない。パラメータの更新は個別に独立して行う事ができるため、パラメータを複数のパラメータサーバに分割することが、容易に可能である。

2.2　SimGrid

SimGrid [1] [9] は、並列計算アプリケーション向けに開発されたシミュレーションフレームワークである。並列計算シミュレータは大別して、エクスペリメンタルプラットフォーム、エミュレータ、ネットワークシミュレータ、アプリケーションシミュレータの四種類があり、SimGrid はアプリケーションシミュレータに属する。

アプリケーションシミュレータの特徴は、シミュレーション時に実際イベントを発生させず、イベントのコストだけを抽出して、シミュレーションを行うことである。このため計算負荷が小さく、単一計算機でも大規模なシステムのシミュレーションを短時間で実行することが可能になる。

SimGrid ではプラットフォーム設定とデプロイメント設定を XML ファイルで記述し、シミュレーションコードを C++で記

```
<?xml version= '1.0 '?>
<!DOCTYPE platform SYSTEM "surfxml.dtd">
<platform version="2">
 <AS id="AS0" routing="Full">
  <host name="host1" power="1E8"/>
  <host name="host2" power="1E8"/>
  <link name="link1" bandwidth="1E6" latency="1E-2" />
  <route src="host1" dst="host2">
    <link:ctn id="link1"/>
  </route>
...
 </AS>
</platform>
```

図 4　SimGrid におけるプラットフォーム記述例

```
<?xml version= '1.0 '?>
<!DOCTYPE platform SYSTEM
"http://simgrid.gforge.inria.fr/simgrid.dtd">
<platform version="3">
 <!-- The master process -->
 <process host="host1" function="master">
  <argument value="10"/><!--argv[1]:#tasks-->
  <argument value="1"/><!--argv[2]:#workers-->
 </process>
 <!-- The workers -->
 <process host="host2" function="worker">
  <argument value="0"/>
 </process>
</platform>
```

図 5　SimGrid におけるデプロイメント記述例

述する。設定例とコード例を示す (注1)。図 4 に簡単なプラットフォームの記述を示す。host1 と host2 が link1 で接続されている簡単な構造である。ホストに対しては計算力が、リンクに対してはバンド幅とレイテンシが定義されている。さらにルーティングとして host1 から host2 へ行くには link1 を通るよう設定されている。

図 5 にデプロイメント設定ファイルを示す。host1 上に master プロセスを、host2 上に worker プロセスを実行しており、その際の引数も与えている。

図 6 にプロセスコードの例を示す。

SimGrid のプロセスは、典型的には、メッセージを受信し、タスクを実行し、メッセージを送信する、というループを実行する。このようなプロセスが多数存在し、それぞれ相互に通信することでシステムとして動作する。

SimGrid の設定ファイルではホストやリングなどの低位の構成要素だけでなく、「クラスタ」のような抽象度の高い構成要素を直接記述できるため、大規模なシステムを表現することが容易である。

3.　耐障害手法の実装

並列システムにおける障害とその復旧に関しては、広く研究が行われている [10]。並列システムにおける障害の種類には、ノード障害、ネットワーク障害など、さまざまな障害が考えられるが、本稿では、ワーカノードの障害のみを考察する。これはワーカノードの台数が大きく、最も障害の可能性が高いと考えられるからである。

本稿ではワーカノードに対して、バックアップノードが存在することを前提とする。ワーカノードに障害が発生すると、バックアップノードを起動し、中断した作業を再開させる。この

(注1)：例は SimGrid のチュートリアルスライド http://simgrid.gforge.inria.fr/tutorials/simgrid-use-101.pdf から抜粋した。

```
int worker(int argc, char *argv[ ]) {
  msg_task_t task; int errcode;
  int id = atoi(argv[1]);
  char mailbox[80];
  sprintf(mailbox,"worker-%d",id);
  while(1) {
    errcode = MSG_task_receive(&task, mailbox);
    xbt_assert(errcode == MSG_OK, "MSG_task_get failed");
    if (!strcmp(MSG_task_get_name(task),"finalize")) {
      MSG_task_destroy(task);
      break;
    }
    XBT INFO("Processing  '%s '", MSG_task_get_name(task));
    MSG_task_execute(task);
    XBT INFO(" '%s ' done", MSG_task_get_name(task));
    MSG_task_destroy(task);
  }
  XBT INFO("I 'm done. See you!");
  return 0;
}
```

図 6　SimGrid におけるプロセスコード例

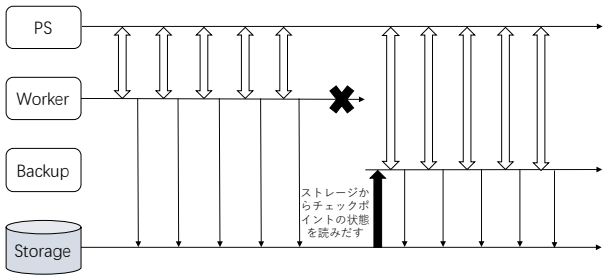

図 7　チェックポイント法 (CKPT 法)

とき、障害を起こしたワーカノードの状態をバックアップノードに引き継ぐ必要がある。本稿では、3 つの耐障害手法を考察する。

3.1　チェックポイントを用いる

チェックポイントを用いる耐障害手法は、多くの並列システムに使われた耐障害手法の一つである。ワーカは一定量の作業を行うと、ワーカの状態をチェックポイントとして記録する。ワーカに障害が発生した場合には、チェックポイントから状態を読みだして再実行することで、処理を続行することができる。この手法を以下 CKPT 法と呼ぶ。

チェックポイントを記録するにはストレージに I/O 操作を行う必要がある。したがって、頻繁にチェックポイントを記録すると実行時間が大幅に増加する。今回実装した手法では、ワーカが学習結果をパラメータサーバにアップロードしたあとに、チェックポイントを取る。

システムは、ワーカの障害発生を検知し、すべてのワーカに障害の発生を通知する。障害が発生したワーカだけではなく、すべてのワーカが実行中の学習を破棄してロールバックし、学習を再実行する。

3.2　パラメータサーバからデータを取得する

一般の並列システムでは情報の重複は少なく、個々のノードの持つ情報に冗長性がない。このためチェックポイントを用いて、情報を複製する必要がある。一方、パラメータサーバを用いた並列機械学習システムでは、同期が終了した時点では、すべてのワーカのパラメータ情報が同一になり、さらに同じ情報がパラメータサーバ群に残されることになる。つまり非常に冗長性が高い。

この手法は、この冗長性を活かして、パラメータサーバを

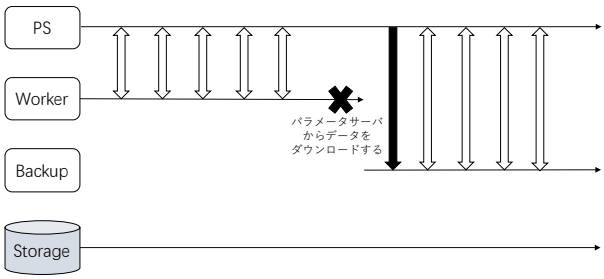

図 8　パラメータサーバからデータを取得する方法 (PS 法)

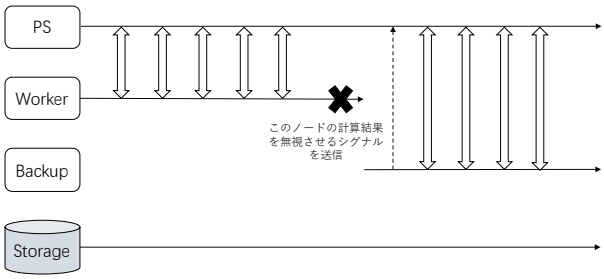

図 9　障害ワーカの結果を破棄する方法 (IGNR 法)

チェックポイントサーバとして利用する方法である。バックアップノードは、パラメータサーバから最新のパラメータ情報を読み取り、処理を続行する。この手法を以下 PS 法と呼ぶ。

具体的には、以下の手順を取る。ワーカが故障すると、システムはバックアップノードを起動する。バックアップノードは、パラメータサーバに最新のパラメータを要求し、ワーカとして実行を再開する。この手法では、直後の同期の際に、障害を起こさなかったノード群が復旧してきたワーカの終了を長く待つことになるが、特に動作を変更する必要はない。

3.3　障害ワーカの結果を破棄する

パラメータサーバは多数のワーカからパラメータの更新情報を受取り、パラメータを更新する。この過程は多分に確率的なので、一回の繰り返しにおいて、1 つのワーカからの更新情報が欠如しても、全体としての学習にはそれほど大きな影響は無いと考えられる。その考えに基づいた手法がこの手法である。

具体的には、パラメータサーバは、ワーカの障害を検知すると、そのワーカからの更新情報を用いずにパラメータの更新を行うように動作する。この間に、システムがバックアップノードを起動する。バックアップノードは、パラメータサーバから最新のパラメータを受領し、ワーカとして復旧する。この手法を以下 IGNR 法と呼ぶ。

前の二種類の手法と比べて、この手法は I/O 操作にコストがかからなく、復旧時間も一番短いと予想されるが、学習の精度が低下する恐れがある。

3.4　故障の発生間隔

計算機の故障発生は、ポアソン分布に従う。ある事象がパラメータ λ のポアソン分布を従う時、その事象の発生間隔はパラメータ λ の指数分布を従う。指数分布の期待値は $1/\lambda$ なので、故障の平均間隔は λ の逆数となる。この平均間隔は MTBF(Mean time between failures) と呼ばれる。本シミュレーションでは、あらかじめワーカノードの MTBF を設定し、指数分布に従って、故障が発生する時刻を決定した。

表 1　ノードとモデルの詳細設定

項目	詳細
学習タスク 1 個の処理時間	100s
学習モデルのサイズ	1GB
チェックポイントストレージの I/O 速度	1GB/s
チェックポイントデータサイズ	8GB
ノードの起動時間	5s
計算機一機につきの MTBF	3Years = 94608000s

表 2　シミュレーションのネット環境

	回線	バンド幅	レイテンシ
ワーカクラスタ	ノード-バックボーン	125MB/s	50μs
	バックボーン	5GB/s	500μs
サーバクラスタ	ノード-バックボーン	5GB/s	500μs
	バックボーン	0.5TB/s	500μs
クラスタ	外部回線	0.5TB/s	500μs

4.　SimGrid を用いたシミュレーション

本研究では、並列環境シミュレータ SimGrid を用いて評価を行う。シミュレータを用いることで、仮想の評価環境を必要に応じて設定することができる。

4.1　故障の実現

SimGrid のプロセスは、データ送信中、データ受信中、処理実行中のいずれかの状態を取る。今回の実験では、時間的に支配的な処理実行中の障害のみを扱った。

処理を実行する際に、指数分布に従って乱数を発生させ、障害までの時間を決定する。予定処理時間中に障害が起きる場合には、障害時間まで処理を行ってから障害を発生させる。予定処理時間中に障害が起きない場合には通常の処理を行う。

4.2　評価環境の構築

今回の評価では、パラメータサーバクラスタが 1 個、ワーカクラスタが 100 個からなるシステムを構築した。システムの構造は図 11 に示す。ノードと、学習モデルの詳細に関しては、表 4.2 に示す。

各クラスタはそれぞれ、100 個のノードを持つ。クラスタ内には、バックボーン回線が存在し、ノードがルータを経由して通信する場合は、まずバックボーンへの通常回線を通り、バックボーン回線を経由してルータに到達する。クラスタ内部の構造を図 10 に示す。システム全体のネットワーク回線の性能設定を表 4.2 に示す。

チェックポイントは、ノードに付随したストレージがあることを仮定し、ネットワークを経由せずに読み出しと書き込みができるものとした。

評価では、3 種類の耐障害手法に対し、それぞれ学習回数 200 回のシミュレーションを 10 回行い、学習の総時間と、障害が発生した回数を記録した。

4.3　評 価 結 果

3 つの手法それぞれの平均学習時間を図 12 に、標準偏差を

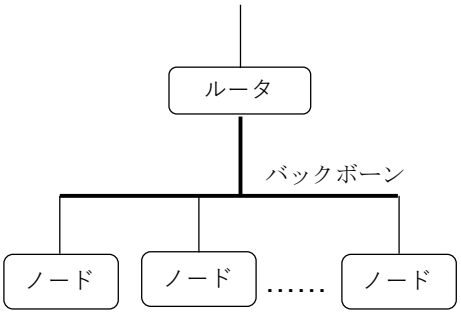

図 10　クラスタの内部デザイン

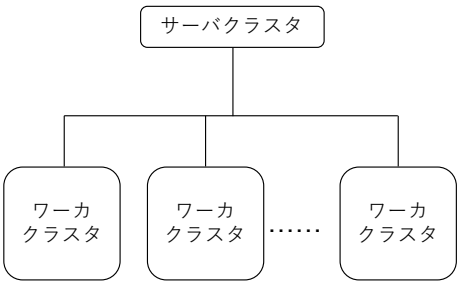

図 11　システム全体のデザイン

表 3　障害が起きた平均回数

手法	CKPT 法	PS 法	IGNR 法
回数	2.5	3.2	3.2

図 13 に示す。また、障害が起きた平均回数を表 4.3 に、障害 1 回あたりの学習全体の平均増加量を表 4.3 に示す。

表 4.3 に示す通り、障害の平均発生回数は約 3 回である。計算時間は約 28700s なので、計算機が評価内で障害を起こす確率は $\lambda = 28700/94608000, k = 1$ のポアソン分布により求められる。計算上の確率は 0.03 %となり、ワーカ台数 10000 台に対して 3 回の故障が発生することが予想されるので、予想通りに障害をシミュレーションできている事がわかる。

図 12 の示す通り、CKPT 法がもっとも計算時間が長い。これは、チェックポイントを書き込むや、読み出す過程に時間を費やしただけではなく、システム全体の再起動も、学習時間の増やす要因になったと考えられる。また、図 13 の示す通り、CKPT 法の標準偏差値も、他二種類の手法と比べて大きくなっている。これは、一回の障害による学習時間の増加が大きいためと、障害が一回の学習中に起こるタイミングが異なるためだと思われる。

表 4.3 の示す通り、IGNR 法は、障害による学習時間に対する影響はほぼ見当たらない。また、障害が発生すると、障害が発生しない場合よりも学習時間は減少する。これは、障害が発生すると、バックアップワーカは障害が発生したことだけをパラメータサーバに知らせるためだと思われる。この動作により、本来学習に費やすべき時間は減り、パラメータの通信コストも無くなる。時間だけ見ると優れている手法だが、学習を破棄するため、システム全体の学習に影響が出る可能性がある。

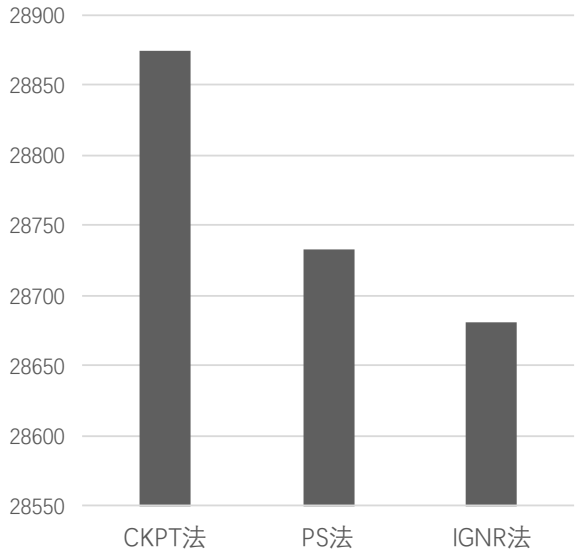

図 12　三つの手法の平均学習時間 (s)

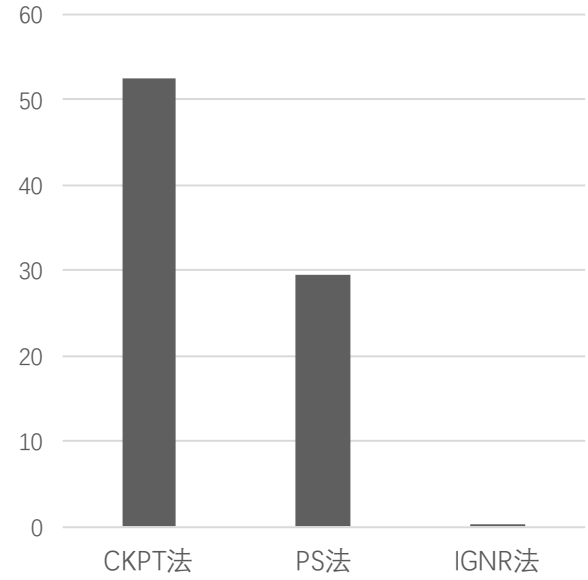

図 13　三つの手法の学習時間の標準偏差 (s)

表 4　障害 1 回につき増加する計算時間

手法	CKPT 法	PS 法	IGNR 法
増加時間 (s)	78.1	12.6	-0.5123

5. おわりに

パラメータサーバを用いた並列機械学習システム固有の対故障性手法について議論するとともに、分散並列環境シミュレータ SimGrid を用いた定量的な評価を行った。パラメータサーバを用いた並列機械学習システムに固有の冗長性を利用した対故障性手法は、一般に広く用いられている定期的なチェックポイントを用いる手法と比較し、コストを大幅に低減できることを示した。

今後の課題は以下のとおりである。

- 本稿では、ワーカノードのみの障害を対象とした。パラメータサーバの障害、ネットワークの障害を考慮することも今後の課題である。
- チェックポイント用のストレージは一般にはグローバルストレージを用いるが、本稿ではローカルにアクセスでき、コンテンションが発生しないことを仮定している。より現実的なチェックポイントストレージ設定でのシミュレーションを行う必要がある。
- 本稿のシミュレーションでは、シミュレーション環境を固定して評価を行った。異なるネットワークなどのさまざまな環境での評価を行う必要がある。
- IGNR 法では、故障したノードの計算を捨ててしまうため、学習が遅延する可能性がある。この遅延を定量的に予測することが今後の課題の一つである。

謝　　辞

この成果の一部は、国立研究開発法人新エネルギー・産業技術総合開発機構 （ＮＥＤＯ）の委託業務の結果得られたものです。本研究は JSPS 科研費 JP16K00116 の助成を受けたものです。

文　　献

[1] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, “Versatile, scalable, and accurate simulation of distributed applications and platforms,” Journal of Parallel and Distributed Computing, vol.74, no.10, pp.2899–2917, June 2014. http://hal.inria.fr/hal-01017319

[2] Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, and A. Ng, “Building high-level features using large scale unsupervised learning,” International Conference in Machine Learning, 2012.

[3] J. Dean, G.S. Corrado, R. Monga, K. Chen, M. Devin, Q.V. Le, M.Z. Mao, Marc ’AurelioRanzato, A. Senior, P. Tucker, K. Yang, A.Y. Ng, “Large scale distributed deep networks,” NIPS 2012: Neural Information Processing Systems, 2012.

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

[5] M. Li, D.G. Andersen, J.W. Park, A.J. Smola, A. Ahmed, V. Josifovski, J. Long, E.J. Shekita, and B.-Y. Su, “Scaling distributed machine learning with the parameter server,” 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pp.583–598, USENIX Association, Broomfield, CO, Oct. 2014. `https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li_mu`

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

[7] 黎　明曦, 谷村勇輔, 一杉裕志, 中田秀基, “マスタ・ワーカ型パラメータサーバの実装と besom への適用,” 信学技報, vol. 115, no. 174, CPSY2015-33, pp.179–184, 2015.

[8] L.G. Valiant, “A bridging model for parallel computation,” Commun. ACM, vol.33, no.8, pp.103–111, Aug. 1990. http://doi.acm.org/10.1145/79173.79181

[9] “Simgrid: Versatile simulation of distributed systems: `http://simgrid.gforge.inria.fr/index.php`”. Accessed: 2016-07-11.

[10] Ifeanyi P.Egwutuoha, D. Levy, B. Selic, and S. Chen, “A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems,” The Journal of Supercomputing, pp.1302–1326, Sept. 2013.