Vol.2018-IOT-40 No.18
2018/3/5

MN-1から学んだこと: 大規模並列深層学習向け計算機基盤の作り方

土井 裕介[1]　鈴木 脩司[1]　福田 圭祐[1]　秋葉 拓哉[1]　渡辺 晃平[2]

概要：深層学習は，大量の計算能力を投入する必要があるタスクである．ChainerMN を用いることで，複数のノードに搭載された複数の GPU を連携させて分散学習が可能となる．これを最大限に活用するため，専用の計算機クラスタ: MN-1 を構築し，計算能力を集約運用している．本稿では，MN-1 の初期の運用から学んだ経験を共有しつつ，大規模並列深層学習向け計算機基盤のあり方について検討を行う．

Lessons Learned from MN-1: How to Make A Computing Infrastructure for Massively Parallel Deep-Learning

Yusuke Doi[1]　Shuji Suzuki[1]　Keisuke Fukuda[1]　Takuya Akiba[1]　Kohei Watanabe[2]

1. Introduction

深層学習は，従来のデータセンターや計算機基盤と比較しても大量の計算機資源を必要とするタスクである．例えば 2011 年時点では画像分類の学習に 5 日ないし 6 日かけていた事例が存在する [1]．特に近年の大規模データセットに対する学習タスクの研究開発において，一試行あたりに必要とされる計算量は拡大する一方であり，効率的な研究開発のためには分散計算などによって大規模な計算を迅速に並列実行できる環境を構築する必要がある．その意味において，深層学習向け計算機基盤は，従来の計算機基盤と比べてもややスーパーコンピュータに近づいた構成であり，例えば似たような構成のスーパーコンピュータを Top500 List 等に見つけることができる．一方で，実務でさまざまなタスクが走る，という意味においては業務用の計算機クラスタの側面も持つ．

MN-1 は，そのような深層学習向け計算機基盤として構成された計算機クラスタの一つである．構築の前提として，組織内におけるさまざまな計算タスクを実行すること，そのために ChainerMN[2] を利用することが当初から決まっていた．一方，ChainerMN の設計や，大規模並列深層学習の方法論は構築開始時に決定的なものが存在せず，手探りの構築となった．なお，本システムの構成・構築は NTT コミュニケーションズ (株)/(株)NTTPC コミュニケーションズが実施した．

本稿では，特に MN-1 の初期運用時の経験からの学び，特に構築時の計画と構築後のチューニングについてまとめる．あわせて，大規模並列深層学習向け計算機基盤の現在の課題についてもまとめる．節 2 では MN-1 に期待される計算負荷の性質と，関連研究について述べる．節 3 では実際に構築した MN-1 の構成について説明する．節 4 では初期運用時に得られた課題について述べ，また，MN-1 の初期運用後にはじめて明確になった課題について節 5 でまとめる．

2. 期待される計算負荷と関連研究

MN-1 では，深層学習の計算タスクを実施する．本節では深層学習の計算負荷の性質について簡単にまとめた上で，アクセラレータとして GPU を用いた他の計算機クラスタについてもまとめる．

深層学習の計算タスクのうちの多くは，データをもとに計算モデルを更新しつづけるタスクである．典型的なラベル付きデータセットに対する更新は，単純化すると以下の

[1] 株式会社 Preferred Networks
〒 100-0004 東京都千代田区 大手町 1 丁目 6-1 大手町ビル 2F
[2] NTT コミュニケーションズ株式会社
〒 100-8019 東京都千代田区内幸町 1 丁目 1 番 6 号

ような計算になる．なお，以下，X: 入力データセット，x: 入力データ (バッチ)，t: x に対応する正解ラベルや報酬など，θ: モデルパラメータ，$f(x;\theta)$: θ によりパラメータ化された関数 (ニューラルネットワーク等で実装)，$L(t,y)$: ロス関数とする．計算の目的は，ロス関数を最小化させるパラメータ θ を発見することである．

(1) 入力データセット X をシャッフルする．
(2) 入力データセットから，一定数のサンプルであるバッチとして，x,t を取り出す．
(3) $y = f(x;\theta)$ を計算する．f の具体例については本稿では扱わない．
(4) $l = L(t,y)$ を計算する．L の具体例については本稿では扱わない．
(5) θ の l に対する勾配を偏微分により求める．個別の演算の微分はフレームワークに組込まれており，これに沿ってここまでの計算グラフを逆方向に辿る．
(6) l を最小化する方向に θ を動かす．
(7) 入力データセットが残っていればステップ 2 に戻る．
(8) 入力データセットがなくなればステップ 1 に戻る．なお，ここまでを 1 エポックとして数え，一般的には一定のエポック数を経過した時点で学習を終了させることが多い．

ここで，ステップ 3 からステップ 5 までは，原則的にバッチレベルでの行列式の計算である．ここでは，並列計算が有効に作用するため，現行のアーキテクチャにおいてはっ GPGPU 上の計算が極めて有効に作用することが知られている．具体的には，例えば ImageNet と呼ばれるデータセットは画像サイズが平均して 490x430x3(RGB) 程度であり，これをバッチ数だけまとめた行列をパラメータに対して計算しロスを求め，これを微分し勾配を求め，パラメータを更新する．画像の総数は百万枚を超える．従って，大量の行列の計算が可能であるアクセラレータが実用的な計算には必須である．

現状，このような計算に向いた計算アクセラレータのうち入手性が高いものは，GPGPU と呼ばれるものであり，特に NVIDIA 製 GPU と CUDA ライブラリ/CUDNN[3] の組み合わせが広く使われる．

ChainerMN の典型的な実行では，データパラレルと呼ばれる並列化手法を取る．これは，入力データセット X をプロセス数 N に対して均等に分割して配布し並列に学習を行う．学習中，ステップ 5 とステップ 6 との間で，Allreduce 処理による勾配の集約を行う．Allreduce 処理は，並列処理されている全てのプロセス間の値に対して特定の集約処理 (この場合は平均) を実施し，その結果を全てのノードに配布する処理であり，Message Passing Interface (MPI) で定義されている．なお，Allreduce にはいくつかの方法が知られているが，MN-1 上では比較的 naive な手法である Ring-Allreduce を想定している．対象ノード間に Ring を構築できれば良いため，プロセスの配置にさえ気をつければ比較的ボトルネックが発生しづらい，という利点が存在する．

以上のような集団通信アルゴリズムの利用において現状入手性が高くソフトウェアが成熟している環境は，Infiniband にもとづくものが中心である．

MN-1 は以上のような ChainerMN の負荷を想定して構成した．詳細な構成は節 3 にて示す．また，MN-1 上の ChainerMN の負荷について，もっとも大規模な実験については [4] に詳しい．

なお，先行事例としてスーパーコンピュータを GPU を主たる計算要素として構築した事例は多数存在する．想定される計算負荷や投下できるコストが異なるため単純に比較はできないが，例えば TSUBAME2.0[5] は，主要な計算力を GPGPU によって実現した最初期のスーパーコンピュータの一つである．2010 年 11 月の Top500 List では，Tianhe-1A をはじめとして NVIDIA GPGPU 製の躍進があり，その後 2017 年 11 月時点でも一定数のスーパーコンピュータが NVIDIA 製 GPU を用いている．

3. システムの構成

本節では，MN-1 の構成について述べる．

全体の構成を図 1 に示す．ネットワーク的にはほぼフラットな構成となっている．主に CPU サーバ群 (10 台)，GPU サーバ群 (128 台)，ストレージシステム (1 式) で構成されており，また通常のイーサネットスイッチ (10Gbps) のほかに，FDR Infiniband スイッチ (56Gbps) が存在する．なお，ノードの設置状況を図 2 に示す．

3.1 GPU ノード

MN-1 の主たる構成要素は GPU ノードである．単一のノードに GPU を多数搭載することで，その規模以下の計算のノード間通信コストを抑えること，また，インターコネクトおよび筐体コスト削減のために，当時市場で入手可能であった筐体のうちもっとも筐体あたりの GPU 数が多いものを選択した．主な構成要素を表 1 に示す．

表 1 GPU ノードの構成

筐体	Supermicro 社製 SYS-4028GR-TR2 (4U)
CPU	Intel® Xeon® E5-2667v4 (3.20GHz) x 2
Memory	256GB
GPU	NVIDIA® TESLA® P100 x 8
Interconnect	Mellanox® ConnectX®-3 Infiniband FDR x 2

GPU ノードは 128 台存在し，各ノードに 8 基の GPU を備えるため，GPU の総数は 1024 となる．なお，現状 128 台のうち半分を実験的にログインして利用し，残り半分をジョブキューによって利用している．

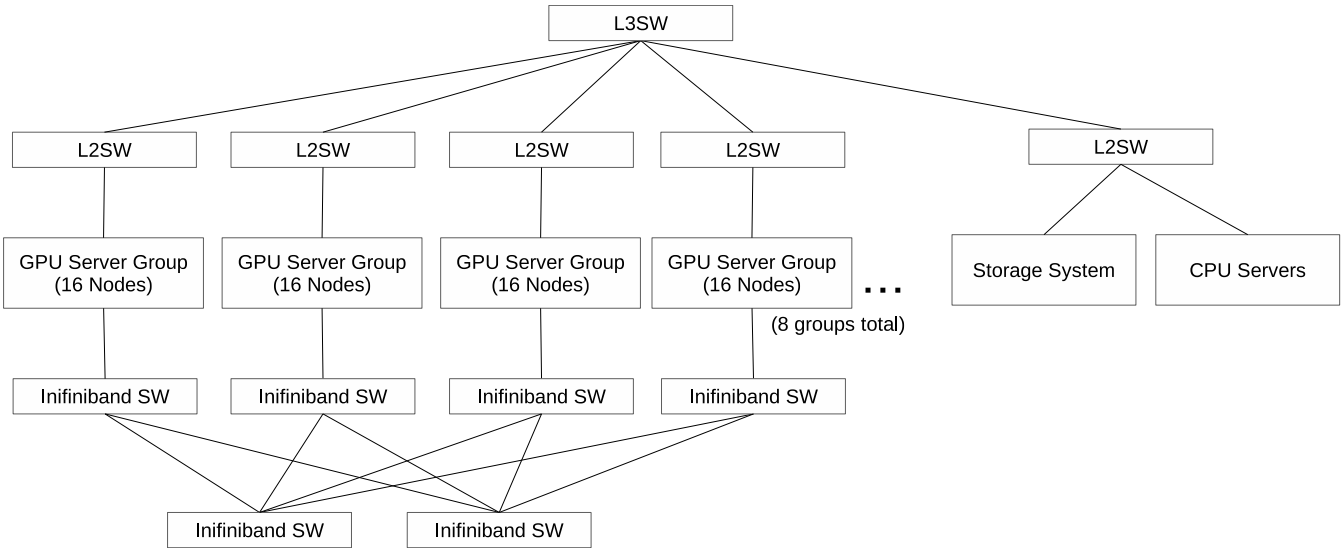

図 1 MN-1 論理構成 (略図)

図 2 MN-1 設置状況

3.2 インターコネクト

図 1 には示していないが，インターコネクト (Infiniband FDR 56Gbps) は冗長化と帯域の確保のために GPU ノード 1 つあたり 2 基づつ用意している．GPU ノード 16 台を 1 グループとし，それぞれのグループに 2 つの Infiniband スイッチを用意し，各ノードからの接続を収容している．このグループを 8 式用意し，それぞれのグループスイッチからの接続を上位のスイッチ 2 基で収容している．

一般にはスーパーコンピュータにおけるインターコネクトはタスクの配置の自由度や計算の自由度を重視して Fat-Tree[6] などの，ノンブロッキングネットワークを構成することが多いが，MN-1 では主にコストの観点から断念している．一方，前述の通り ChainerMN においては並列計算した結果 (パラメータ勾配) の集約に，同期型の Ring Allreduce を採用しているため，分散計算を行うタスクの配置を工夫すればボトルネックは発生しない．また未検証ではあるが，グループをまたぐ複数の並列学習タスクが存在したとしても，同期型の集約を行っている以上計算時間と通信時間は分離せざるを得ず，バースト的な通信が主である．従って，学習パフォーマンスへの影響は限定的であると考えている．

3.3 CPU ノード

MN-1 には GPU ノードに加えて 10 台の通常の IA サーバ (CPU ノード) を持つ．これらはログインサーバや IPMI 管理サーバ，ジョブキュー管理サーバ等に使われている．また，これに加えて 1 台のサーバが，物理メディアによるデータ受け渡し専用サーバとして用意されている．これらサーバは一般的な構成であることから，詳細は割愛する．

4. 初期運用時の課題

本節では，初期運用時に発生した課題について記述する．初期運用時には，いわゆるグランドチャレンジと性能検証を兼ねて，全 GPU マシンを利用しての LINPACK ベンチマークの実行と，ResNet50[7] を用いた分散深層学習のタイムチャレンジを実行した．

4.1 LINPACK の実行と初期運用時のできごと

LINPACK ベンチマークにもちいる HPL(High Performance LINPACK) は，大規模行列式の求解問題であり，解の範囲をタイル状に区切って各プロセスにわりあてて分散計算を行う．このとき，プロセスの数によって実行可能な問題のサイズが変化する．一般に問題のサイズがおおきいほうが計算効率が高いことから，個別のプロセッサ (この場合は GPU) のメモリに載る問題サイズを求め，タイルのサイズを決めるパラメータ P, Q や他のパラメータを変更しながら最適なパラメータを求め，パフォーマンスの測定を行う．

なお、実行されるプロセス数は $P \times Q$ になるため，稼動するプロセス数のきりが悪いと実行効率を上げることが困難になる．今回は 1024GPU を用意したが，1 つでも故障すると全 GPU を利用した実験が困難になってしまう．適切な量の予備資材の確保が必要である．

また，陽な故障であれば対処が用意であるが，発見に時間がかかったのはなぜか PCIe が x16 で認識されるべきところを x8 で認識される，などといった，一見動いているが性能が出ない，といった問題である．これらの問題を早期に発見するには，個別の機器コンポーネントの性能ベンチマークを実施する，という手段も有効だろう．

なお，いわゆる DGEMM，つまり倍精度の行列のかけ算によるベンチマークを一部のノードにおいて実行した結果を図 3 に示す．なお，NVIDIA のデータシートによると，理論最高性能は倍精度で 4.7TFLOPS である．ここに見えるように，同じ GPU カード，筐体であっても計算能力に大きな差があることがわかる．これが GPU カードの個体差なのか，温度の問題なのか，一種の相性問題なのかは不明である．LINPACK を含めた同期型分散計算はいちばん遅いノードにより律速されてしまうため，極めて遅いカードについては別のカードと交換して，可能な範囲で最悪値を改善した．

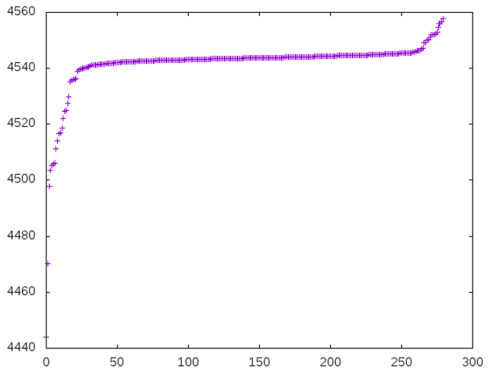

図 3 DGEMM による性能のばらつき (一部の GPU のみ): 縦軸は性能 (GFLOPS) であり，横軸は性能順にソートした．

HPL の実行は以下のポリシで実施した．

- 小さい規模からおおきい規模へ徐々にスケールアップしながら最適化
- パラメータ探索はベイズ最適化により実施

P, Q のみならず，HPL の実行効率に関係するパラメータ空間がおおきいことから，大規模での最適化は困難である．従って，1GPU，1 ノード (8GPU) から開始して，徐々に大規模の実験を実施した．小さい規模の実験は早く終えることができるため，パラメータサーチが効率的に行える．パラメータサーチは，事前に定めた範囲での組み合わせ最適化を行う Hyperopt[8] により実施した．

1 ノードでは効率 72%程度で動作していたが，通信が増えれば増えるほどインターコネクトのボトルネックが効いたのか，最終的には効率 28%(1.39PFLOPS) であった．インターコネクトについては調査したが，SNMP レベルでモニタできる帯域としては十分に余っていることが判明した．一方，トラフィックがバースト的に (同期的に) 発生していると期待されることから，短期的にキューイングが発生して計算機がその間待つ必要があったのではないかと推測している．

4.2 ImageNet in 15min 実験

あわせて，ImageNet データセットの，ChainerMN を用いた ResNet-50 による 90 エポックの分散学習を実施した [4]．これは，[9] の設定を再現した上で，規模の向上のために学習方法 (特にパラメータ更新の手法) を改善したものである．なお，本実験ではあくまで学習の速度を測定するために，ImageNet データセットについては事前に各 GPU ノードのローカルストレージにコピーして利用し，パス名のみの通信により各ノードが学習データをロードする構造になっている．

この実験では，ノード間通信に NVIDIA 社製の NCCL2[*1] を利用した．これはノード間通信も含めた集団通信アルゴリズムの実装であり，NVIDIA 社製 GPU と Infiniband HBA との間の通信を含めて最適化されている．実際にはこの規模の実験ではいくつかの細かい問題があり，ここに一部を共有する．

まず，NCCL2 の当時のバージョンにおいては，プロセス数が多くなる場合にいくつかの注意が必要であった．NCCL2 の各プロセスにおいて，プロセス数に応じたファイルディスクリプタを必要とし，また，あわせてスタックも大量に消費する．従って，一定以上の規模 (MN-1 の環境においては 800 プロセス程度) になると OS の保護機構によりプロセスが停止し，ulimit コマンド等で制限を解除して動作させる必要があった．さらに，処理時間の都合でタイムアウトするノードが発生していたため，MPI Barrier により初期化後に同期してから次のステップに進める必要があった．

また，通信がボトルネックとなることがわかっていたため，ここでは通信トポロジを環境変数経由で具体的に指定する必要があった．Ring Allreduce を用いることで帯域的なボトルネックの問題はなくなったが，そのためには通信トポロジを陽に指定する必要がある．また，Ring Allreduce はノード間通信時間が全体の性能にダイレクトに影響する．通信量を抑えるために，計算は単精度浮動小数点数 (fp32) で実行し，通信のみ半精度浮動小数点数 (fp16) で行った．

図 4 に，今回の実験で計測した ChainerMN のスケーラビリティを示す．図中には学習中の各イテレーションにかかった時間と，その中でも特に通信にかかった時間が示されている．256GPU まではよくスケールしているが，512GPU 以上になるとかかる時間が増大している．

[*1] https://developer.nvidia.com/nccl

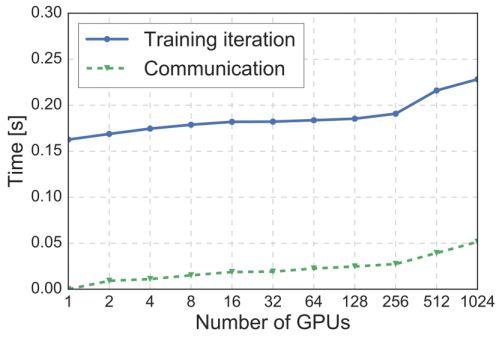

図 4 ChainerMN のスケーラビリティ

5. 今後の課題

本稿では，大規模分散深層学習を実行するための基盤である MN-1 について，特に運用初期の事例について報告した．その後業務での活用が開始されており，大規模な実験を実施することは難しい状況であるが，ここまでにわかったこと，またその後業務利用で判明した事実も含めて今後の課題として最後にまとめる．

節 4.2 で述べたとおり，大規模分散深層学習のベンチマークにおいてはデータセットのロードは計測範囲外であり，そのため事前に全てのデータを全てのノードに配布している．しかし，当然のことながら業務においては多様なデータセットが存在し，これをそのタスクに割り当てられた GPU ノードに，定められた形式 (HDF5 形式や `npz` など，Chainer が扱いやすい形式) で円滑に配布することが必要である．例えば [10] にあるように，計算力を活用するためのワークフローシステムに，データの前処理と供給の部分を組込む必要があると考えている．また，そのためには，現状 Chainer では通常のファイルシステムしか想定していない部分を，例えばオブジェクトストレージ API の利用や，Hadoop のようなシステムとの連携なども含めた拡張を想定したい．実際に，初期の段階ではナイーブにイメージファイルをサンプル毎に open していたのを，データセットをアーカイブにして fseek で対応するだけで性能が向上している．一方，例えば思ったよりも CPU 処理のみで完結する (GPU を活用できない) ジョブが多く，CPU のみのクラスタを別途作る，などといった物理的な改修も検討している．

深層学習の研究開発はサイクルが早い．一方，設備計画はこの規模になると短くとも 1 年程度は必要である．従って，設備構築・インフラ運用と，その上で利用するソフトウェアフレームワークの開発，利用者の教育 (best current practice の迅速な共有や最適化) などは，その時間サイクルのズレを考慮した上で進める必要がある．

参考文献

[1] Krizhevsky, A., Sutskever, I. and Hinton, G. E.: Imagenet classification with deep convolutional neural networks, *Advances in neural information processing systems*, pp. 1097–1105 (2012).
[2] Akiba, T., Fukuda, K. and Suzuki, S.: ChainerMN: scalable distributed deep learning framework, arXiv preprint arXiv:1710.11351 (2017).
[3] Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B. and Shelhamer, E.: cudnn: Efficient primitives for deep learning, arXiv preprint arXiv:1410.0759 (2014).
[4] Akiba, T., Suzuki, S. and Fukuda, K.: Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes, *arXiv preprint arXiv:1711.04325* (2017).
[5] Matsuoka, S., Endo, T., Maruyama, N., Sato, H., Shin ’ichiroTakizawa：The total picture of tsubame2. 0, *TSUBAME e-Science Journal, GSIC, Tokyo Institute of Technology*, Vol. 1, pp. 2–4 (2010).
[6] Leiserson, C. E.: Fat-trees: universal networks for hardware-efficient supercomputing, *IEEE transactions on Computers*, Vol. 100, No. 10, pp. 892–901 (1985).
[7] He, K., Zhang, X., Ren, S. and Sun, J.: Deep residual learning for image recognition, *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778 (2016).
[8] Bergstra, J., Yamins, D. and Cox, D. D.: Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms, *Proceedings of the 12th Python in Science Conference*, pp. 13–20 (2013).
[9] Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y. and He, K.: Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, *arXiv preprint arXiv:1706.02677* (2017).
[10] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. et al.: TensorFlow: A System for Large-Scale Machine Learning., *OSDI*, Vol. 16, pp. 265–283 (2016).