Vol.2017-OS-139 No.5
2017/3/1

異種AIエンジン統合クラウドの実現に向けたシステムソフトウェアFlow OSの構想

高野 了成[1,a)]　池上 努[1]　須崎 有康[1]　田中 哲[1]　広渕 崇宏[1]

概要：エネルギー効率の高い人工知能処理基盤を実現するため、用途ごとに異種のプロセッサを動的に組み合わせる異種 AI エンジン統合クラウド Flow in Cloud (FiC) と、FiC 上の資源管理を行うオペレーティングシステム Flow OS の構想について述べる。FiC は GPU や FPGA といった AI エンジンをサーキットスイッチネットワークを介して直結し、アプリケーションの各処理ステージごとに最適な資源を割り当てることが可能である。Flow OS は FiC の資源管理を担うとともに、利用されていない資源の電源管理を行うことでデータセンタ全体のエネルギー効率を高める。ExpEther と Bare-Metal Container を用いた FiC 模擬環境の構築と同環境上での Flow OS の試作についても紹介する。

1. ポストムーア時代のコンピューティングシステム

2030 年にはデータセンタ内部で処理されるデータ量は現在の 10 倍以上に達すると予想される。データセンタにおける処理の中でも今後急増すると見込まれるのが、深層学習や機械学習に代表される人工知能処理である。特に深層学習のトレーニングフェーズはいかに多くのデータを利用するかに性能が大きく依存するため、データ量はさらに上昇修正される可能性がある。また、ムーアの法則の終焉が近づくにつれ、半導体の集積度や速度向上が限界を迎え、汎用 PC サーバを多数並べてスケールアウトさせる現状のデータセンタアーキテクチャは、電力的にも性能的にも早晩限界に達する。

まず深層学習処理で用いられる演算ハードウェア（以下、「AI エンジン」と記す。）に着目して背景を詳解する。多数の PC サーバを接続したクラスタ型計算機上での分散学習手法の研究開発に関して、当初は Google の DistBelief [1]（TensorFlow [2] の前身）や Microsoft の Project Adam [3] など、汎用 CPU を用いた処理が主流であった。その後、CUDA 等 GPU を用いたソフトウェアスタックの成熟に伴い、Parameter Server [4] など、GPU クラスタ型計算機を効率的に用いた分散学習への移行が始まった。また、FPGA は、汎用な計算資源として利用可能な段階まで技術は成熟していないが、深層学習の推論や学習の一部を GPU よりも電力効率よく処理できるとの報告がある。これらに加えて、深層学習では一般的な科学技術計算と比べて必ずしも高い演算精度は必要でないため、Eyeriss [5]、Google Tensor Processing Unit、Intel Nervana Engine のように機械学習処理の一部を専用 ASIC として実現することで省電力化・高性能化を実現する試みも増えている。また、人間の脳の動きを模倣した IBM TrueNorth などのニューロモーフィックチップ、組合せ最適化問題に特化した D-Wave Systems の量子アニーリングマシンなど、新しい計算パラダイムや原理を活用したプロセッサの研究開発も行われている。以上、上記で言及した AI エンジンは処理内容によって性能や電力効率の面で一長一短がある。

また、三次元積層技術や光ネットワークによるインタコネクトの広帯域化によって演算ハードウェア間のデータ移動のコストが劇的に低減される。例えば、広域網で利用されている波長多重や多値変調をデータセンタ内ネットワークに適用することで、ファイバあたり数 10 Tbps 級の帯域が 2030 年頃には実現可能であるという試算が示されている [6][7]。

我々は、このような用途特化型演算ハードウェアと超広帯域通信を組み合わせた新しいコンピューティングパラダイムである「フローセントリックコンピューティング」を提案し [8]、その実証に向けて異種 AI エンジン統合クラウド（**Flow in Cloud**）の開発に着手した。フローセントリックコンピューティングでは、データのある場所で処理を行うという従来の「データアフィニティ型」処理に加えて、計算機能が存在する場所にデータを積極的に移動して処理を行う「ファンクションアフィニティ型」処理を導入

[1] 国立研究開発法人 産業技術総合研究所 情報技術研究部門
Information Technology Research Institute, National Institute of Advanced Industrial Science and Technology (AIST)
[a)] takano-ryousei@aist.go.jp

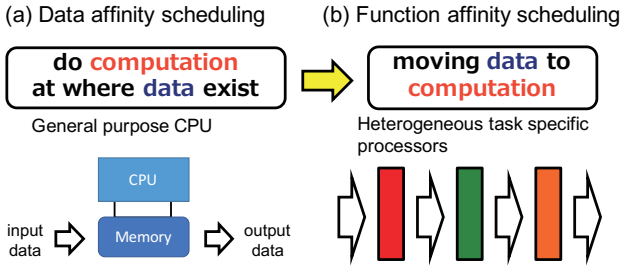

図 1 データアフィニティ処理とファンクションアフィニティ処理

することで、演算ハードウェアと広帯域通信の能力を活用する（**図 1**）。さらにシステムソフトウェアの観点からは、システム全体の資源利用効率や性能を最適化するシステム化技術の創出、及びそれを具現化したデータセンタワイドなオペレーティングシステムの実現が課題となる。我々は、異種の AI エンジンが連なるフローに着目して資源管理を行う点から、このオペレーティングシステムを **Flow OS** と呼ぶ。

本論文の構成は次のとおりである。2 節では、異種 AI エンジン統合クラウド Flow in Cloud (FiC) の概要について、ハードウェア、システムソフトウェア、プログラミングモデルの各々の観点から述べる。3 節では、ExpEther を用いた FiC の模擬環境と Flow OS の試作について述べる。最後に 4 節でまとめと今後の予定について言及する。

2. Flow in Cloud: 異種 AI エンジン統合クラウド

2.1 ハードウェア構成

既存のシステムでは、GPU や FPGA は PCI バス経由でホスト PC と接続されるアクセラレータの形態を取っているため、デバイス間で直接データ交換を行うことは基本的に考慮されておらず、ホスト PC のメインメモリを介した通信になる。NVIDIA は GPU 間のインタコネクト技術として NVLink を提案しているが、あくまでノード内での規模の Point-to-Point 接続技術でありスケーラビリティは考慮されていない。我々は、PCI バスやホスト PC を経由するオーバーヘッドを除去し、多数の異種 AI エンジンの柔軟な組み合わせを実現するために、AI エンジンを搭載した基板 (AI エンジンノード) と、それらをつなぐネットワークから構成される Flow in Cloud (FiC) と呼ぶ異種 AI エンジン統合クラウドを開発している。FiC は、計算機仮想化技術を用いることなく、必要な AI エンジンハードウェアを組み合わせた実行環境（スライス）を利用者に提供する、ベアメタルクラウドの一形態と捉えることができる。

FiC の概要を**図 2** に示す。AI エンジンノードは、GPU や FPGA などの AI エンジンに加え、DRAM、ネットワーク I/O、制御用 CPU を有す。制御用 CPU は演算を実行するものではなく、あくまで GPU や FPGA の設定を行うための補助的なもので十分である。AI エンジン間のネットワークは複数のスイッチノードから構成され、将来の光ネットワークの導入も見越して、パケットスイッチではなく、サーキットスイッチネットワークを前提とする。光ネットワークの場合は、光の速度のままでスイッチングを行える利点があるが、電気ネットワークの場合でも実装が簡単になるという利点もある。一方、通信の開始前に AI エンジンノードの端点同士で接続を確立する必要があることを意味し、これを効率的に利用するには、後述する Flow OS といったシステムソフトウェアとの連携が必要になる。スイッチノードは FPGA、DRAM、複数リンクのネットワーク I/O を持ち、FPGA 上にサーキットスイッチ機能が実装される。

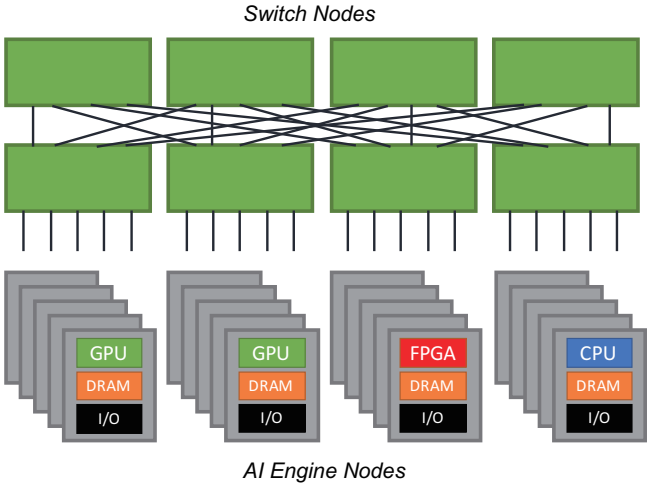

図 2 Flow in Cloud (FiC) のハードウェア概要

FiC と既存提案（Intel Rack Scale Architecture (RSA)、HP The Machine、NVIDIA DGX-1）の比較を**表 1** にまとめる。Intel RSA は基本的には既存ハードウェア技術の延長であり、サーバ単位ではなくラック単位で資源管理を行う点を主張している。HP The Machine はユニバーサルメモリを中心とする共有メモリ型並列計算機であり、共有メモリがボトルネックになるため大規模なシステムは想定されていないと推測する。一方、FiC はスケーラブルな拡張が可能であるが、システムソフトウェアを開発する必要がある。この点について次節以降で述べる。

2.2 Flow OS

上記で述べた FiC のハードウェア資源を管理・制御し、アプリケーションに応じた実行環境を提供するオペレーティングシステムを Flow OS と呼ぶ。**図 3** に Flow OS の概要を示す。Flow OS は、FiC の資源である AI エンジンノードとスイッチノードを統一的に管理し、利用者に対して、論理的なアプリケーション実行環境であるスライスを提供する。スライスに対する資源の割当てやスケジューリングも Flow OS が担うが、データ処理フレームワークご

表 1　Flow in Cloud と既存提案の比較

	Intel RSA	HP The Machine	NVIDIA DGX-1	Flow in Cloud (提案)
アーキテクチャ	資源プール（CPU、ストレージ、IO）から従来型サーバを動的に構成して提供	ユニバーサルメモリを中心とする共有メモリ型並列計算機	高性能 GPU を実装したサーバ内の複数 GPU 間を高速なリンク NVLink で接続	異種 AI エンジンによる適材適所の処理＋分散細粒度学習制御
利点	資源のプール化による利用効率、柔軟性の向上・既存ソフトが利用可	複数の SoC が共有メモリを介して通信することで、通信の自由度が高く、既存ソフトを利用可	GPU で高速化しやすい処理に最適化	異種デバイス活用した飛躍的な効率アップ＋常に進化するシステムを実現
欠点	既存のサーバベースなので、非線形的（劇的）な効率向上が困難	スケーラビリティに課題。ユニバーサルメモリデバイス（メモリスタなど）の実用性が不明瞭	GPU に向かない処理の高速化とサーバ間通信に課題	異種複数エンジンの管理や、既存ソフトの利用のために OS、ミドルウェアで工夫が必要
スケーラビリティ	— サーバ間は従来型通信技術を利用	△ 共有メモリが隘路となり、台数に限界	× NVLink でつなぐのは筐体内のみ	◎ 分散制御により、大量の AI エンジン実装が可能
電力効率	△ 個々のサーバへのリソース割り当ての最適化のみ	? ユニバーサルメモリの構成に依存。低消費電力化には課題	× GPU は基本プロセッサベースで、消費電力大	◎ 処理内容に応じて電力効率の良い AI エンジンに各タスクを自動割り当て
データ転送	— サーバ間は従来型通信技術を利用	○ ユニバーサルメモリ共有可能台数内では高効率	○ 単一サーバ内は高速。サーバ間は従来型通信技術を利用	◎ AI エンジン間での分散学習制御により、データ転送効率化

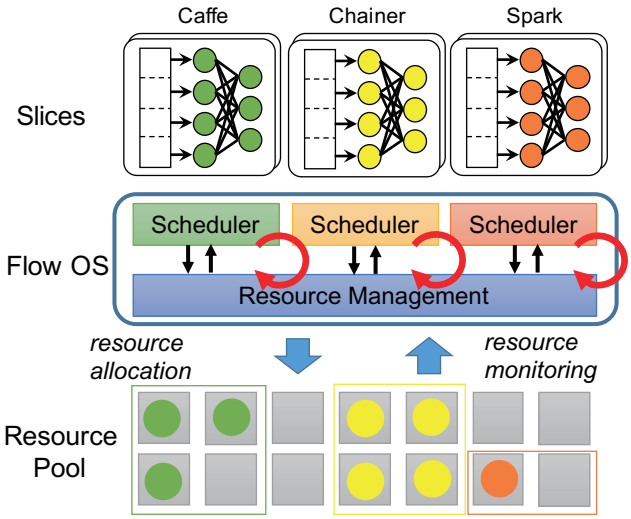

図 3　Flow OS の概要

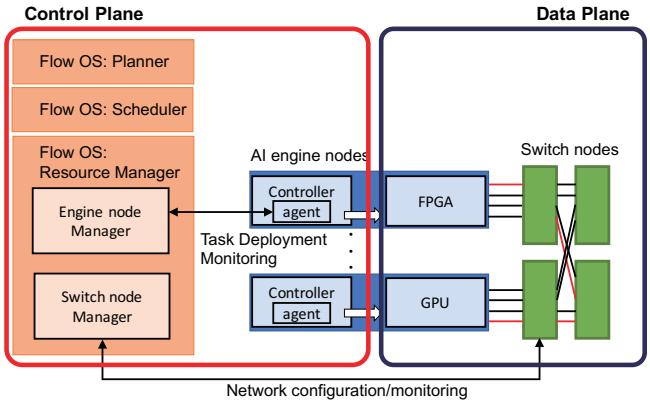

図 4　Flow OS の設計：制御プレーンとデータプレーンの分離

とにスケジューリングポリシを設定することができる。

既存のクラスタ管理ミドルウェアは、各計算ノード上に Linux 等フル機能の OS が動作することを前提としている。一方、FiC では、アプリケーションの実行効率を追求するために、AI エンジンがデータ処理に集中するために、極力外部からの干渉を防ぐ必要がある。そこで**図 4** に示すように、OS の構造を制御プレーンとデータプレーンに分離し、性能と機能性の両立を図るという着想に至った。データプレーンは、アプリケーション実行に特化した環境であり、ホスト CPU を介さず AI エンジン間で直接データの交換が可能である。一方、制御プレーンは、資源管理、ユーザ管理に必要な機能性を提供し、Flow OS と各 AI エンジンノードに搭載された制御用 CPU 上で動作する Flow OS agent が連携することで動作する。

Flow OS の主な機能は下記の 2 点である。

(1) スライスの提供：ユーザからの要求に応じて、アプリケーションの実行に必要な種類と数の AI エンジンをプランニングし、資源プールから AI エンジンを確保しスライス（論理的な計算機）を構築する。さらに、各タスクが最適な AI エンジン上で動作するように、プランニング自身も機械学習（強化学習等）により最適化する手法について

検討する。なお、スライスは一般の OS におけるプロセスに相当する概念である。スライスは複数のタスクから構成され、各タスクは AI エンジンに割り当てられ動作する。

プランニングに際しては、下記の点に配慮する必要がある。

- FPGA の再構成可能なデバイスではあるが、ビットストリームの転送から再構成までを含めると CPU におけるプログラムロードと同程度のレイテンシは期待できない。したがって、各 AI エンジンにどのようなプログラムがデプロイされているかも含めて、スケジューリングする必要がある。
- ネットワークトポロジを考慮して、レイテンシの少ない経路を選択する。
- スイッチノードに搭載された FPGA を用いたインネットワークプロセッシング（集約などの簡単な処理を想定）を考慮する。

(2) スライスのライフサイクル管理：AI エンジンへのプログラムのロード、実行、監視を実行する。スライスのモニタリング結果を反映して、AI エンジンの利用率が最大化されるように、AI エンジン使用数等を常時調整する。スケジューリングポリシはジョブの種類によって選択が可能である。

2.3 プログラミングモデル

FiC 上のアプリケーションは複数の AI エンジンでの処理（以降、「flowlet」と記す。）を組み合わせて記述する。flowlet は計算とデータ転送の集合として抽象化できるものとし、タスクを頂点、フローを辺とした有向非循環グラフ（Directed acyclic graph; DAG）で定義する。

Flowlet の定義を以下に記す。(1) タスクは 0 個以上の入力ポート及び出力ポートを持つ。(2) フローは出力ポートと入力ポートを接続する。(3) ひとつの出力ポートから複数の入力ポートへ接続できる。すなわちマルチキャスト通信が可能である。(4) ひとつの入力ポートへはひとつの出力ポートからしか接続できない。(5) タスクの実行は、すべての入力ポートからのデータを受け取ってなんらかの処理を行い、すべての出力ポートにデータを出力する。

以下、バイト単位での加算（`AddBytewise`）をふたつ実行して、そのふたつの結果を連結（`Concat`）する Flowlet のプログラム例を用いてプログラミングモデルを説明する。なお、本プログラムの DAG グラフを**図 5** に示す。図中の箱がタスク、タスクを結ぶ線がフロー、フローの横の数字が入出力ポート番号である。AI エンジン上で実行されるタスクの実装方法について、ここでは規定していないが、OpenCL や CUDA、Vivado HLS 等を用いて実装することを想定している。

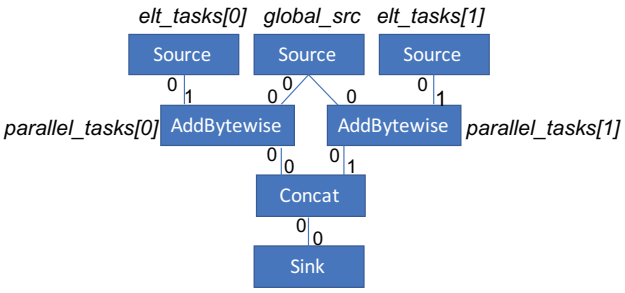

図 5 プログラム例：グラフの頂点がタスク、辺がフロー、 辺の両端の数字が入出力ポート番号を示す。例えば、`Source` の出力ポート 0 が `AddBytewise` の入力ポート 1 に接続されている。

`Source.new`、`AddBytewise.new` など[*1]によりタスクが生成される（3、8、10、13、14 行目）。引数として可変数の出力ポートを取り、i 番目の引数に与えた出力ポートは生成したタスクの入力ポート i 番へ接続される。なお、引数に出力ポートでなくタスク自体が与えられると、そのタスクの出力ポート 0 番として扱われる。`Task.execute` により当該タスクを起点にフローをたどって DAG を構築し、まずスライスをセットアップする。続いて DAG をトポロジカルソートなどで、その依存関係を解析し、タスク群を順次実行する（16 行目）。

```
1  n = ArrayData.length
2
3  global_src = Source.new(GlobalData)
4
5  elt_tasks = []
6  parallel_tasks = []
7  n.times {|i|
8    elt_src = Source.new(ArrayData[i])
9    elt_tasks << elt_src
10   parallel_tasks << AddBytewise.new(global_src,
       elt_src)
11 }
12
13 concat = Concat.new(*parallel_tasks)
14 sink = Sink.new(concat)
15
16 sink.execute
17 p sink.result
```

本プログラムでは明示的にどのタスクをどの AI エンジンノードで実行するか指定していないが、このプランニングとスケジューリングを行うことが Flow OS の役割である。例えば、マスタ・ワーカ処理において、100 個のワーカタスクの結果をマスタで集約するような DAG の場合、AI エンジンノードが 100 個確保できるとは限らない。そ

[*1] これらは `Task` クラスの派生クラスである。

の場合は、与えられたノード数を用いて実行できるように時間的または空間的なスケジューリングが必要になる。そして、Flow OS はタスクを AI エンジンノードにデプロイし、ノード間通信を可能にするためにスイッチノードの設定を行う。

3. Flow OS の試作

3.1 ExpEther を用いた FiC 模擬環境の構築

まず、2.1 節で述べた FiC のハードウェアと並行して Flow OS の開発を進めるための模擬環境を構築した。ここでは、AI エンジンノードの代わりに PCI デバイスを用い、アプリケーションの要求に応じて、ホスト PC と PCI デバイスの接続を動的に変更する方針とした。例えば、GPU を利用するアプリケーションに対しては GPU クラスタをスライスとして提供し、GPU は不要だが高速なストレージが必要なアプリケーションに対しては NVMe を搭載したクラスタをスライスとして提供する。このようにホスト PC と PCI デバイスをディスアグリゲーションする技術として、ExpEther [9][10] を採用した。ExpEther は PCI Express バスを一般的な Ethernet 上に拡張できる PCI Express over Ethernet 技術であり、ホスト PC は ExpEther ホストバスアダプタを利用して、PCI Express 規格準拠デバイスをネットワーク越しに結合することが可能になる。

論文執筆時の模擬環境では、40 ギガビット Ethernet スイッチを介して、2 台の PC サーバと、複数台の NVIDIA P100 GPU、及び Intel NVMe SSD が接続されている。

3.2 Flow OS 試作システム

2.2 節で述べた通り、Flow OS の役割であるスライスの提供とライフサイクル管理を、既存のソフトウェアを適用することで模擬環境上で試作する。前者の実現には、Apache Mesos [11]、後者の実現には、Bare-Metal Container (BMC) [12] を利用した。

3.2.1 Apache Mesos

Apache Mesos はクラスタ資源管理ミドルウェアであり、クラスタ内の資源を統一的に管理する資源マネージャ、及び MPI や Spark 等のワークロードが異なるフレームワークごとにモジュール化された（アプリケーション）スケジューラから成る 2 レベルスケジューリングを採用している。また、**図 6** (a) に示すように、オファーベースのスケジューリングモデルを採用しており、クラスタの状態のサブセットのみをスケジューラに公開する点が特徴的である。

各計算ノードにはエージェントが動作し、利用可能な資源（CPU やメモリ、ディスク等）をオファー（offer）する。マスタは各エージェントのオファーを集約し、さらに各フレームワークに通知する。フレームワークは必要な資源のオファーが来たら、それを受理（accept）し、計算ノードの Executor を介してタスクを実行させる。なお、丸四角で示したものがオファーのメッセージであり、エージェント ID と利用可能な資源のリストが含まれる。

Mesos は ExpEther のように、計算ノードと GPU 等の資源が分離され、動的に追加・削除することは想定していない。また、後述するが、BMC が想定するように電源断した計算ノードは管理対象外となる。したがって利用していない資源にも通電しておく必要があり、電力効率を高くすることができない。そこで、エージェントとマスタ間にプロキシエージェントを配置し、ExpEther によるデバイスの接続操作や BMC の処理を隠蔽するように拡張する。

拡張方法の概要を図 6 (b) に示す。まずオファー時には、プロキシエージェントは各エージェントがオファーした資源に加え、ExpEther 経由で利用可能な資源をオファーメッセージに追加する。オファーメッセージの改変方法には、図で例示したように 1 ノードあたりのデバイス数を最大化する楽観的な方式や、実際のデバイス数と同様の数しかオファーしない悲観的な方式など、複数のポリシを選択する余地がある。前者では先にオファーを受理したフレームワークは資源を確保できるが、それ以降のフレームワークではオファーの受理に失敗することになる。また、フレームワークがその資源を受理したときに、計算ノードが電源断している場合は、BMC を用いて計算ノードを起動し、続いて ExpEther API を用いて計算ノードと PCI デバイスの接続確立を行う。

Mesos が採用するオファーベースのスケジューリング方式は、FiC の要件とミスマッチしていないかに議論の余地がある。オファーベースの問題点として、information hiding や hoarding の問題が知られている [13]。hoarding 問題とは、MPI やステートフルなストリーム処理を実行する場合にはあらかじめ必要な資源が揃うまでオファーを待つ必要があるが、その間中途半端な資源が蓄積されたままの状態に資源の利用効率が低下する。これらの問題に対して、中央集権的なモデルや、Google Omega [13] のような Shared-state モデルが代替案として検討できる。

3.2.2 Bare-Metal Container

Mesos によって構築されたスライス上に OS カーネルやアプリケーション実行環境をデプロイする必要がある。そこで、本試作では Bare-Metal Container (BMC) [12] を利用する。Docker 等のコンテナは、アプリケーション実行環境の可搬性の点では有効な技術であるが、仮想マシンとは異なり、利用者が OS カーネルを選択することはできない。BMC は Docker コンテナとネットワークブート、リモート電源制御を組み合わせた技術であり、アプリケーションに最適化された Docker コンテナとカーネルを任意に組み合わせてデプロイ・起動できる。また、利用されていない資源を電源断し、利用時のみに通電することも可能である。

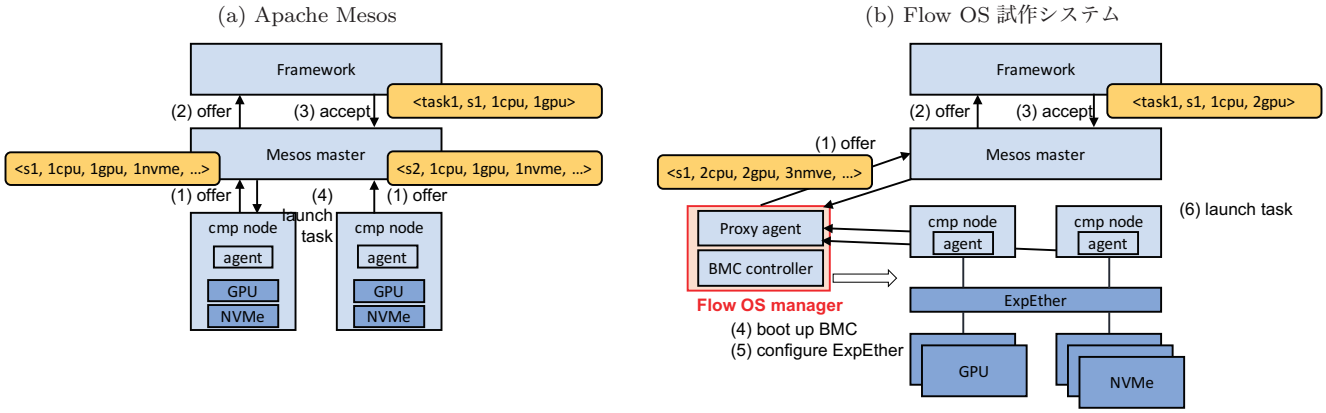

図 6 ExpEther を用いた FiC 模擬環境における試作システムの概要と動作の流れ

4. まとめと今後の予定

本論文では、人工知能アプリケーションの実行ステージごとに異種のプロセッサを適材適所で組み合わせることができるベアメタルクラウド FiC のオペレーティングシステムである Flow OS の構想について述べた。異種の AI エンジンを多数組み合わせたシステムの可能性を引き出すためには、アプリケーションやアルゴリズムに対して、これらの資源を、ネットワークやストレージまで含めて、どのように組み合わせるべきか、システム全体の構成を最適に制御するアーキテクチャが必要であり、このような議論や開発は世界的にも端緒についたところである。

我々は、FiC のハードウェアの開発と並行して、汎用 PC と ExpEther を用いた模擬環境を構築し、それを用いて Flow OS の試作システムを開発中である。本試作では、スライスを管理するための資源管理システム及びスケジューラとして Apache Mesos を、実行環境のデプロイ機構として Bare-Metal Container を用いている。

現時点では試作が完了していないため、詳細な設計や性能評価については別途報告する予定である。今後は試作を完成させ、Flow OS 及び FiC ハードウェアへ要件の洗い出しとフィードバックを行う予定である。

謝辞 本研究は、NEDO 委託事業「IoT 推進のための横断技術開発／省電力 AI エンジンと異種エンジン統合クラウドによる人工知能プラットフォーム」での研究成果の一部を用いている。また、FiC のハードウェア開発を担当し、定期的に議論して頂いた東京大学 工藤知宏教授、慶應義塾大学 天野英晴教授、国立情報学研究所 鯉渕道紘准教授に大いに感謝する。

参考文献

[1] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V. et al.: Large scale distributed deep networks, *Advances in Neural Information Processing Systems (NIPS)*, pp. 1223–1231 (2012).

[2] 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, *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (2016).

[3] Chilimbi, T. M., Suzue, Y., Apacible, J. and Kalyanaraman, K.: Project Adam: Building an Efficient and Scalable Deep Learning Training System., *USENIX OSDI*, Vol. 14, pp. 571–582 (2014).

[4] Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J. and Su, B.-Y.: Scaling Distributed Machine Learning with the Parameter Server., *USENIX OSDI*, Vol. 14, pp. 583–598 (2014).

[5] Chen, Yu-Hsin and Krishna, Tushar and Emer, Joel and Sze, Vivienne: Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks, *IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers*, pp. 262–263 (2016).

[6] Kudoh, T., Ishii, K. and Namiki, S.: Current status and future prospects of optical communications technology and possible impact on future BDEC ssytems, *4th Big Data Extreme Computing Workshop 2016* (2016).

[7] Inoue, T., Kurosu, T., Ishii, K., Kuwatsuka, H. and Namiki, S.: Exabit Optical Network Based on Optical Comb Distribution for High-Performance Datacenters: Challenges and Strategies, *Frontiers in Optics 2015 OSA Technical Digest, FTh3C.3* (2015).

[8] Takano, R. and Kudoh, T.: Flow-centric computing leveraged by photonic circuit switching for the post-moore era, *10th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pp. 1–4 (2016).

[9] Suzuki, J., Hidaka, Y., Higuchi, J., Yoshikawa, T. and Iwata, A.: ExpressEther - Ethernet-Based Virtualization Technology for Reconfigurable Hardware Platform, *14th IEEE Symposium on High-Performance Interconnects (HOTI)*, pp. 45–51 (2006).

[10] ExpEther Consortium: http://www.expether.org (2016).

[11] Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R. H., Shenker, S. and Stoica, I.: Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center., *NSDI*, Vol. 11, pp. 22–22 (2011).

[12] Suzaki, K., Koie, H. and Takano, R.: Bare Metal Container, *18th IEEE International Conference on High Performance Computing and Communications (HPCC)*, pp. 25–36 (2016).

[13] Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M. and Wilkes, J.: Omega: flexible, scalable schedulers for large compute clusters, *8th ACM European Conference on Computer Systems (EuroSys)*, ACM, pp. 351–364 (2013).