Vol.2022-HPC-185 No.2
2022/7/27

マルチ FPGA・GPGPU 混在環境向け資源管理システムの提案

LI YANZHI[1]　菅谷みどり [1]

概要：人工知能を実現するため高度な計算力が必要とされている．こうした計算力を達成するため，様々なアクセラレータが提案されている．しかし，アクセラレータごとに，支援が可能なアルゴリズムや，支援方法が異なることから，アルゴリズムによっては高速化が難しいケースが存在する．異なる種類のアクセラレータを使うことで，より性能を向上させせることができると考えられる．しかし，現在，そうした仕組みは十分に提案されていない．また，複数のアクセラレータを統一的に利用できる汎用的なインターフェイスや，障害へ対応する仕組みは提供されていない．そこで，本研究は異なるアクセラレータを持つマルチ FPGA・GPGPU が混在している環境を中心に，アクセラレータの特徴に応じて計算資源を管理するミドルウェアを提案する．提案したミドルウェアは FPGA・GPGPU などの異なるクセラレータを同一的に扱うためのインターフェースを提供し，かつ，アクセラレータの故障に柔軟な対応できるものとした．評価では，複数のユーザの同時アクセス時の計算資源の割り当て，かつ障害を模擬した環境で実験を行い，有効性を示した．

キーワード：FPGA，GPGPU，ミドルウェア，ヘテロジニアス計算，JSON-RPC

1. はじめに

近年，人工知能などの技術により，IoT デバイスなどから収集した膨大なデータの分析や，分析結果を新しいサービスにつなげ，結果を人にフィードバックし，共有することで有効活用する社会が提案されている[1]．こうした社会においては，人間のデータ分析能力を超えた大量のデータと人工知能を生かして，これまで達成できなかった価値を社会にもたらすことが期待されている．大量のデータを近距離で処理する技術として Edge Computing サーバが提案されている[2]．人工知能に代表される複雑で計算量の大きい処理は，ネットワークに接続した小規模なデバイス上のマイコンや，小規模なロボットなどの組込みシステム上で実現することは困難である．このことから，近距離で大規模性能を持つ Edge Computing サーバは，その性能向上技術が期待されている[3][4][5]．特に近年では，第５世代移動通信技術の発展に従い，デバイス側で負荷が大きい計算を，大規模計算を得意とするクラウドにオフロードするなどの処理が可能になっている[6]．このことから，ロボットなどのアプリケーションでは，人工知能などの計算を Edge Computing サーバにオフロードすることで，性能改善を行うことが期待されている[7].

Kyoung-Su らは，人工知能が求める計算は，マルチプロセシング環境において，効率的な実装ができないという課題があることを指摘した．Kyoung-Su らは，CPU を用いた場合には，多くの時間を要する問題があるとしている[8]．これに対し，FPGA をサーバ処理に用いる研究が提案されている[9]．三宮らは，複数のロボットの特徴をモデル化する演算のために，汎用 CPU が実装されたサーバの計算力を利用した．汎用 CPU では，期待されている処理速度の達成が難しい結果だったため，アクセラレータを用いた計算の高速化を行った．その結果，汎用 CPU に対して FPGA での実装は性能を 70 倍に向上することに成功した[9]．しかし，単体の FPGA のみの処理の提案であり，サーバに適した大量の処理能力は提供していない制限があった．

これに対し，天野らは，FPGA を搭載したサーバを複数台利用するための計算クラスタとして FiC を提案した[10]. FiC はネットワークを経由し複数の FPGA を利用できることから，大量の FPGA の計算力を提供することに適している．しかし FPGA が搭載されたサーバを複数ユーザが同時に利用するには，複数のユーザが同時に利用するための仕組みが必要である．具体的には，複数の資源利用者の管理および，利用対象となる FPGA の排他制御を行うための管理機能が必要となる．しかし，初期の FiC にはこうした仕組みは提供されていなかった．これに対して，山倉らは，FiC の管理のための FiC-RM(Resource Manager)を提案し，複数台サーバに搭載した FPGA に対する管理機能を提供した[11]. 山倉らが提案したシステムでは，サーバに搭載された FPGA をユーザに利用可能とした．一方，FPGA チップの情報はユーザに提供しなかったことから異なる FPGA チップを同時に利用できないという課題があった[11]．さら

1 芝浦工業大学
Shibaura Institute of Technology, 3-7-5 Toyosu, Koto-ku, Tokyo 135-8548, Japan

ⓒ2022 Information Processing Society of Japan

に全ての FPGA が接続している前提で FiC-RM に書き込まれたため，サーバの障害などにうまく対応できないという課題があった．このように，異なる FPGA チップを同時に利用できないという課題，障害への対応不足，という２つの課題により十分な利用が困難である課題があった．

FPGA で高速化できるアルゴリズムとして，三宮らが行った線形回帰の多変量解析などがある．一方，大量のデータに対して同じ処理を行う場合には，FPGA はデータスループットが低いという制限がある [12]．また，行列演算を行う場合には，GPU の方が適している．このことから，アルゴリズムによってサーバの計算資源を使い分け出来るのが望ましい[13]．異なるアクセラレータは各自異なる特徴があり，それが得意とするアルゴリズムも異なる．サーバ側の計算性能を利用したいアプリケーション開発者にとっては，Edge Computing サーバ側が，どのようなアクセラレータであっても，汎用的にプログラムを実行できることが望ましい．しかし，現状は，アプリケーションごとに適するアクセラレータや開発ツールを準備する必要がある．また，複数種類のアルゴリズムを高速化するため，複数種類のアクセラレータを利用するための仕組みは十分提案されていない．

異なる種類のアクセラレータを搭載するシステムとしてヘテロジニアスシステムが提案されている[14]．こうしたヘテロジニアスシステムにおいては，FPGA と GPU を共に利用するのみならず，低消費電力で高性能計算が実現できることに期待されている[13][14]．しかし異種類のアクセラレータが混在しているヘテロジニアス環境では，アルゴリズムによって自動的にハードウェアを選べない課題がある．また， FPGA を載せたサーバクラスタにおいては，異なる FPGA チップにおける BitStream 互換性の問題などの課題がある．さらに，計算資源の割り当ての仕組みも提案されているが，固定的なブロックの割り当てのみで，ジョブを中心とした資源管理などが十分に行われていない課題がある[15]．

本研究ではこれらの課題に着目し，高性能計算を可能とする Edge Computing システムの実現を目指し，

(1) 複数のアクセラレータの透過的な利用
(2) FPGA クラスタの有効活用
(3) アクセラレータ障害の隠蔽

の実現を目的とした．目的を達成するために，MEC-RM (Multi-Access Edge Computing Resource Manager) を提案する．MEC-RM は，

(1) 複数のアクセラレータを透過的に利用するための高レベルの抽象インターフェースの提供
(2) FiC-RM の管理機構として排他制御管理機構の提供
(3) 計算サーバの接続情報管理のため障害隠蔽機構の提供

を行う． MEC-RM は，アクセラレータの差異を隠蔽することにより，異なるアクセラレータを統一的に利用できない課題を解決する．また，計算サーバの接続情報を管理しつつ，計算資源の割り当て仕組みを提供する．このことの実現により，障害に柔軟な対応ができない課題も解決する．本論文では，これらの提案機能の設計と実装および予備評価について述べる．我々の知る限り，FPGA と GPGPU の計算クラスタを透過に扱うためのヘテロジニアスシステムは提案されておらず，新規性がある．また，本研究の評価では，MEC-RM によりジョブの割り当てが行われ，かつ，障害に柔軟な対応ができていることを確認し，有効性を示すことができた．

本論文の構成は，次のとおりである．まず，2 節にて，既存アクセラレータの仕組みについて議論を行い，3 節にて提案システムの議論を行う．4 節にて，システムの評価を行い，5 節にて結論とした．

2. 先行研究と課題

2.1 処理速度の向上を目指した研究

既に CPU のクロック数は上限に迫り，単体での性能向上の限界に達した．これらに対して，さらなる計算性能の向上を目指し，様々なアクセラレータの研究がなされてきた[8][12][13][14]． アクセラレータは，特定の計算処理速度を向上させる専用ハードウェアである．既に様々なハードウェアが開発されてきたが，GPGPU(General-Purpose Computing on Graphics Processing Units) が画像処理以外の目的でも応用され普及している[16]．また，FPGA も省電力等の特性から主に組込みシステムなどに利用されている[10]．

2.1.1 GPU

GPU は，グラフィック演算装置として設計された．コンピュータ上の画像はピクセルの組み合わせである．グラフィック演算はこれらのピクセルの値を計算することである．ピクセルは独立して計算できるため，グラフィックス演算は並列可能である．同一画像の各ピクセルは同じアルゴリズムで計算することが多いため，多くの GPU は一つの指令で複数のデータが処理できる SIMD アーキテクチャで実装されている[17]．SIMD アーキテクチャは，一つのプログラムカウンタに対して複数の演算ユニットが配置されることが，特徴とされている．本設計により，同じ面積に CPU より演算ユニットが多く配置でき，同じ消費電力でより高い計算力が達成できる．

ピクセルに対する演算は，行列演算と類似していることから，GPU で行列演算を行う研究が進んでいる[18]．こうしたグラフィックス演算以外にも使う GPU は，GPGPU と呼ばれ， AI 計算に広く利用されている．また，ソフトウエアにおいても，Torch[19]や TensorFlow[20]など GPU を用いた AI 計算のためのライブラリが数多く提案されている．しかし，分岐が多い複雑な計算に対しては十分な仕組みが提供されておらず，少ないデータに対して複雑な計算を実現したい場合には，FPGA などと組み合わせて動作させることが期待される[12][13].

2.1.2 FPGA

FPGA はプログラム可能な集積回路である. アルゴリズムに応じて専用の電子回路を構成することが可能である. FPGA で最適化された回路設計を実装すれば, 計算が何倍も速くなると同時に省電力を実現することができる.

本来, FPGA は専用集積回路を設計するため使われていた. しかし, 専用集積回路の生産やテストなどは時間がかかるため, 最近では, FPGA をそのまま専用集積回路としても使用するようになっている. FPGA は専用回路を構成できるため命令デコーディングなどの回路が不要で, 低消費電力での高性能計算が期待されている. 特に暗号やエンコーディングなどのアルゴリズムは, FPGA での高性能な実装ができる利点がある.

2.2 FiC システムと M-KUBOS

2.2.1 FiC システム

近年, FPGA を汎用的な計算資源として, サーバに複数搭載し, 利用させる手法が提案されている[10][11]. 中でも, 複数の FPGA を回線交換ネットワークによって接続し, 計算を高速に実行する Flow-in-Cloud（FiC）システムが提案されている（図 1）[10]. FPGA 間のデータはホスト PC を介さず FPGA 間で直接データ交換できるので, システムの資源利用率の向上が期待できる.

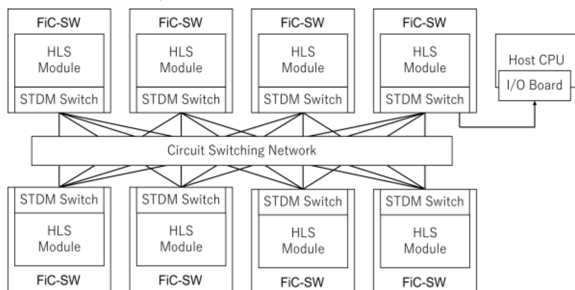

図 1 FiC システム図

図 1 に示したように, FiC システムは複数枚の FiC-SW ボートで構成されている. FiC-SW は Multi-Access Edge Computing (MEC) 環境を想定し開発されたものである. FiC-SW は高速回線交換ネットワークによって接続されていることから, FPGA クラスタを構成できる. FPGA の再構成や回線交換ネットワークの設定で, クラスタの動作を設定できるようになっている.

2.2.2 FiC-SW

FiC-SW ボードは, Xilinx 社が開発した FPGA チップを利用し, ラズパイ Raspberry Pi 3 で FPGA 制御するサーバ基板である. Raspberry Pi 3 で FPGA を構成し, 回線交換ネットワークの動作を指定できる. Raspberry Pi と FPGA 間のデータ伝送もできる. しかし, データ伝送は Raspberry Pi の GPIO を使っているため, 伝送速度が低い問題がある.

2.2.3 M-KUBOS

天野らは, FiC-SW におけるデータ伝送効率問題を解決するため, M-KUBOS ボードを開発した[11]. M-KUBOS は Zynq Ultrascale+ MPSoC チップと 4GB の DDR4 SDRAM を持つサーバ基板である. Zynq Ultrascale+ MPSoC は Xilinx 社が提供した SoC の一種であり, ARM プロセッサと FPGA の両方を搭載している. ARM 部を Processing System (PS), FPGA 部を Programmable Logic (PL) と呼び, PL 部を PS 部から制御可能である. PS 部と PL 部が同じメモリをアクセスできるため, 従来型（FiC-SW）が存在したデータ伝送問題が解決されている.

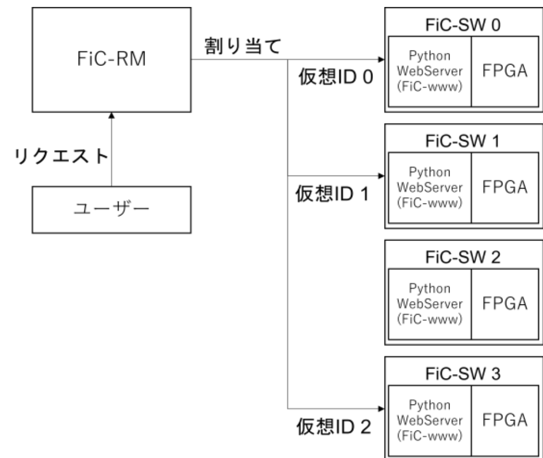

図 2 FiC-RM の動作

2.2.4 利用環境

前節で紹介した FPGA クラスタの利用については, 二つのツールが提供されている. それは FPGA プログラムを開発する Vivado と各ボードを制御する FiC-www である. Vivado は Xilinx 社が提供した開発ツールである. 特に Vivado HLS というツールがあり, C 言語で FPGA を開発できる. FiC-www はクラスタにある各ボードに実装された制御ツールである. HTTP インターフェースを介し, FPGA や回線交換ネットワークを再構成できる. クラスタユーザーが必要に応じて, そのインタフェースで FPGA と配線交換ネットワークを再構成し, データのやり取りを行う[11].

2.3 FiC の課題と関連研究

FiC システムを Edge Computing サーバの計算資源として利用しようとした場合, 以下のような問題点が存在する.

一つ目の問題は, アクセラレータを排他的に利用できないことである. 現状では, ユーザがアクセラレータを利用するためには, アクセラレータがあるサーバに直接アクセスし, アクセラレータを直接操作する必要がある. ユーザ自身が勝手にアクセラレータを搭載したサーバにアクセスし, 他のユーザが使用中のアクセラレータを操作可能である. そのため, 複数のユーザーが同時にこのシステムを利用する際に, 衝突が発生するという問題がある.

二つ目の問題は, 異なるアクセラレータを汎用的に利用

ⓒ2022 Information Processing Society of Japan

できないことである. FiC システムが互換性のない二種類の FPGA チップによって構成され, ユーザがこのことを常に意識したプログラムを記述する必要があることである．利用している FPGA チップが違うため，同じプログラムに対しても生成した BitStream が違っている. 間違った BitStream を FPGA に書き込んでしまった場合, 正しい計算が困難となり, ボードを止まらせてしまう問題がある. このことから, 開発者は常に利用しているボードを意識する必要がある．さらに, 将来 GPU など別種類のアクセラレータを導入する場合，本問題はより重要な課題となる.

三つ目の問題は，障害に十分に対応できない課題である．FPGA などのアクセラレータはバグのある実装に障害が発生しやすい問題がある．それは，FPGA の開発周期が長く，バグが見つかりにくいことが原因の一つである. FPGA の障害発生時には，その FPGA を載せたサーバも接続できなくなり，計算サービスを提供できなくなる．そのため，発生した障害に柔軟な対応が必要である.

山倉らは一つ目の問題を解決するため，FiC-RM (FiC-Resource Manager)というシステムを開発した．FiC-RM は FPGA クラスタに存在するすべてのボードとサーバを管理するプログラムである．本プログラムは, 接続されている複数の FPGA ボード全ての情報を管理し，ユーザーの要求に応じて，サーバを割り当てる処理を行う．また, 割り当てたボードに対し, サーバの番号を管理し，どこのサーバにボードが割り当てられても，同じ番号で制御できるようにした.

図 2 は．ユーザが FPGA の搭載されたサーバを３つ利用するリクエストを送付した際の FiC-RM の稼働例である. クラスタに存在するサーバ情報は前もって FiC-RM に書き込まれている．FiC-RM は自分が持つサーバ情報を参照し，各サーバに状況確認のリクエストを出す．サーバ情報確認後, 未使用サーバに仮想 ID を作成し，ユーザに割り当てる．ユーザは，その後，仮想 ID でサーバに搭載した FPGA の再構成やデータのやり取りを行う.

しかし，山倉らのシステムは，実際には FiC-SW 一つのサーバしか制御できない. また, 先に述べた３つの課題の内, 残りの２つの課題については対応してない. 例えば, ユーザが FPGA クラスタを利用するためには，FPGA の BitStream を, サーバにアップロードする必要がある．しかし，現状ではユーザがアップロードした BitStream ファイルは, FiC-SW と M-KUBOS の両方に対応できないことから，課題２にある互換性のないボードを同時利用できない問題がある．また，FiC-RM は FiC-SW の状況を管理していないため，課題３にある故障に柔軟な対応ができない問題が残っている.

3. 提案

3.1 概要

本研究では, 高性能計算を行う Edge Computing システムにおいて，複数のアクセラレータを透過的に利用できる仕組みの実現を目的としている．目的の実現のためには，2 節に述べた FPGA クラスタの有効活用とアクセラレータ障害の隠蔽を実現する必要がある.

このことから，本研究で提案を行う管理システム(Resource Manager; RM) は，まず，マルチ FPGA におけるボードの排他利用を可能とするため，サーバ側でボードの利用状態をまとめて管理する仕組みを提案する．このことにより，３つ目の課題である障害の隠蔽を行うこともできる．また，RM は複数アクセラレータの透過的利用のために，ユーザが利用しやすいインタフェースとして, JSON-RPC を用いた高レベル遠隔手続呼び出しインタフェースをユーザに提供する. この方法で互換性のないボードを同一のインタフェースで操作でき，2.3 節に述べた二つ目の課題を解決できるという利点がある

3.2 全ての計算資源を管理するプログラム

一つ目の問題である，ボートを排他的に利用できない問題については, 山倉らの解決案が適切だと判断する. 山倉らの解決案はクラスタに存在するすべてのボードを一つのプログラムにより管理することである．この管理プログラムは, 計算資源の利用申請を受け取り, 確保できる計算資源を確認しつつ割り当てる．このことで, 管理プログラム排他制御を行うことができることから，一つ目の課題を解決する.

3.3 高レベルな計算インタフェース

二つ目の課題において，プログラマは異なるアクセラレータごとにプログラムを作成する必要がある問題がある．本来アプリケーションを設計する際には, システムは，ハードウェア特徴を隠蔽することで，開発者の負担を減らす必要がある．本研究では，この問題を解決するために，システムに高レベルな計算インターフェースを設計・実装した.

既存システムではシステムを利用するアプリケーションの開発者は，利用したいハードウェアに直接アクセスし，BitStream の導入や，データのやり取りをプログラムに記述する必要があった. 本提案において, 高レベルな計算インタフェースを導入することにより，開発者はハードウェアにアクセスすることは一切せずに目的を達成できるようにした.

本提案における高レベルな計算インタフェースでは，例えば「画像から顔を認識せよ」など具体的な処理まで定義すれば, アプリケーションは処理したいデータと, 必要な計算プログラムを呼び出すことで計算資源を利用することができる用途を想定した．本方式によりハードウェアへの直接アクセスを隠蔽し，二つ目の課題を解決できるようにした.

ⓒ2022 Information Processing Society of Japan

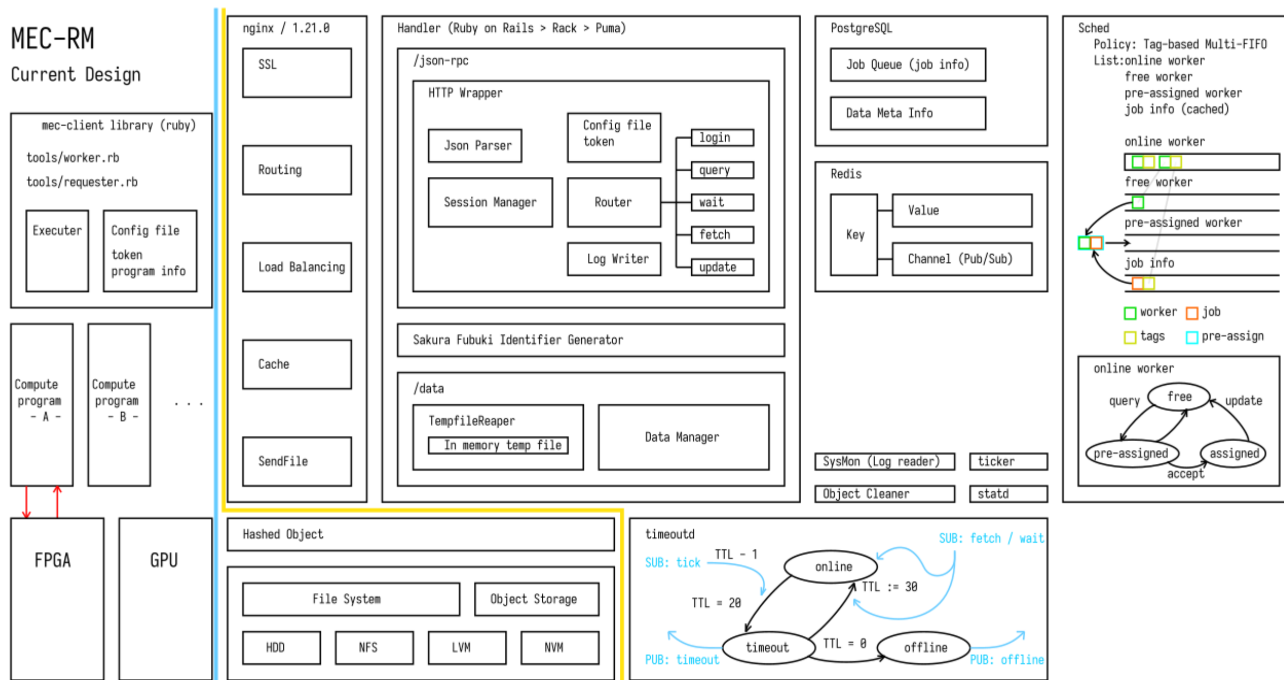

図 3　MEC-RM の構成

3.4　動的な接続情報管理

FiC-RM の三番目の課題である故障に対応できない理由としては，FiC-RM の仕組みとして事前に指定した接続情報を参照して資源割り当てを行う方式にある．事前に指定した接続情報を参照して資源を確保するため，対象となるボードに故障が発生していたり，構成が変わったりしている場合に，対応できず，問題となる．

本問題の解決にあたり，本研究では管理プログラム側には接続情報を持たせず，計算資源を提供しているボードが，情報参照の要求時に動的に自分の情報を管理プログラムに提出する方法を提案する．さらに，ボード自身が定期的に管理プログラムにハートビート情報を送信し，故障がないことを示す設計とした．本実装により管理プログラムが故障に柔軟に対応でき，問題を解決する．

4.　MEC-RM の設計と実装

4.1　システムの基本概念

提案システム全体の概要を図 4 に示した．

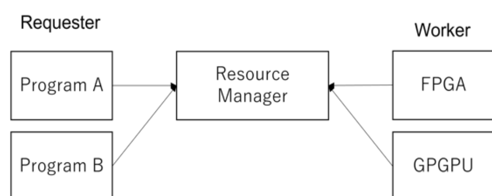

図 4 提案システム概要

提案システムにおいては，複数のアクセラレータを透過的に利用できる仕組みを実現するために，基本的な処理単位に，論理的な命名を行った．まずこれらの用語について以下説明する．

1. 計算資源提供者（ワーカー）：システムで利用可能なアクセラレータを対象とする．自身の計算資源をシステムに提供する論理的な計算単位とする．
2. 計算資源利用者（リクエスター）：システムに接続し，システムに存在する計算資源を利用する利用者を単位とする．
3. 計算資源管理者（マネージャー）：計算資源の提供者(ワーカー)と，利用者（リクエスター）の間に位置し，計算資源を割り当てるものとする．
4. ジョブ：ワーカーが実行できる仕事の最小単位である．通常には入出力データ，実行すべきプログラムとプログラム渡すパラメータが含まれている．
5. パイプライン：一連のジョブによって構成するもの．
6. クライアント：計算資源提供者と利用者の総称．
7. メッセージ：クライアントがマネージャーとやり取りをするとき，相互に送信するもの．
8. ハートビート：接続が継続していることを確認するためのメッセージ．主にクライアントとマネージャーの間で，長時間にわたってメッセージが送信されてない時に送信する特別なメッセージ．

4.2　MEC-RM の概要

本システムの設計・実装を図 3 に示した．本システムはいくつかのコンポーネントに分けられる．各パーツは独立して動作し，複数マシンにも分散可能である．本実装は，C 言語を中心に行った．全部のコードは約 9000 行で，C 言語の部分は約 8000 行である．なお，スケジュール機能の一部は組込み Ruby によって実装した．

ⓒ2022 Information Processing Society of Japan

表 1　Redis で利用した Key

メソッド	名前	内容
GET/SET	session.SESSION.name	このセッションに関連付けられたワーカーの ID
GET/SET	session.SESSION.role	このセッションに関連付けられたクライアントのロール
GET/SET	session.SESSION.ids	このセッションに使った RPC メッセージの ID
GET/SET	worker.WID.session	このワーカーに関連付けられたセッション ID
GET/SET	worker.WID.assign	このワーカーに割り当てたジョブ ID
GET/SET	worker.WID.tags	このワーカーに定義したタグ
GET/SET	job.JID.info	ジョブの情報
GET/SET	job.JID.status	ジョブの実行状況
PUB/SUB	wait	リクエスターが待っているジョブの ID（待っていないジョブでは一旦中止）
PUB/SUB	fetch	ワーカーがジョブの割り当てを請求する（請求しないワーカーに割り当て中止）
PUB/SUB	timeoutd	タイムアウトしたセッション ID
PUB/SUB	jstatus	実行状態が変わったジョブ
PUB/SUB	query	請求されるジョブ ID
PUB/SUB	assign	ワーカーID とこのワーカーに割り当てたジョブ ID
PUB/SUB	offline	オフラインセッション ID

以降，各コンポーネントとその動作を説明する．

4.2.1 通信プロトコル

リクエスターの代表である組込みシステムでは，計算資源が限られ，プログラムをアップデートする機会も少ない．また，ネットワーク接続も不安定であり，ファイアウォールにブロックされる場合もある．通信プロトコル設計にはこれらの状況を十分に考慮すべきである．

こうした事情を考え，本システムにおいては，マネージャーとクライアント間は HTTP に基づいた通信プロトコルを利用する．これは，組込みシステムの不安定なネットワーク接続に対応することを想定している．長い接続を持たず，計算資源への要求を出す時に限りリクエストをだすものとした．また，ファイアウォール事情を考慮し，本来アプリケーション層である HTTP をトランスポート層として利用する．さらに，HTTP のコンテンツとして JSON-RPC に基づいた通信プロトコルを載せる形式とした．組込みシステムではアップデート機会は限られる.そのため,誤った動作から，システムを保護する必要がある．これらの理由により，スキーマとデータ両方も含んだ JSON-RPC プロトコルを選んだ．

さらに，本実装においては，JSON-RPC でバイナリデータを伝送するには不効率と考え，直接 HTTP を利用して伝送するように実装した．計算の入出力データなどは，HTTP POST リクエストでアップロードし，HTTP GET リクエストでダウンロードを可能とした．

4.2.2 フロントエンド：nginx

MEC-RM は nginx[21]を HTTP フロントエンドとして利用している．nginx がクライアントからの接続を受け取り，必要に応じて SSL もここで処理する．SSL 処理が終了したらリクエストをバックエンドに伝送する．

バックエンドサーバから戻ったレスポンスに対しては，必要に応じて SendFile ヘッダーがある．SendFile ヘッダーには伝送すべきファイルのパスが記述してある．nginx は，このヘッダーの検出後，Hashed Object プールからファイルを取り出し，クライアントに送る．本設計によってバックエンドサーバの負荷を減らし，より多くの接続が可能となる．

4.2.3 データ保存：Hashed Object と PostgreSQL

Hashed Object はバイナリデータを保存する場所である．データを SHA256 でハッシュを計算し，ハッシュ値をファイル名としてデータを保存する．本手法で，同じデータに対して繰り返しアップロードした場合であっても，実際に保存されるのは初めの一個のファイルのみとなる．このことから，二つ目以降は効率的に既にアップロードしたものを利用することができる．

また，各ファイルには ID を振った．クライアントがデータをサーバからダウンロードしたりサーバへアップロードしたりする時はこの ID を使う．この時，ファイルは複数の ID と関連付けられる．この関連情報は PostgreSQL データベースに保存される．

また，ストレージの使用が拡大しないように，空間回収プログラムを実装した．プログラムは PostgreSQL にアクセスし，一定期間内使われていないファイルを特定し，削除する．データアップロードやダウンロード，ジョブでデータを入出力データとして参照する際は，このファイルを期間内に設定し，回収対象から外す．

4.2.4 キャッシュサーバ：Redis サーバ

本実装では，各コンポーネント間の共有キャッシュとして Redis サーバを利用した[22]．Redis サーバでは，二種類の共有キャッシュを利用している．一つは GET/SET メソッド

を利用した Key-Value 型キャッシュ，もう一つは PUB/SUB メソッドを利用したメッセージングキャッシュである．利用している Key とチャンネルの一部，その使い方を表 1 で示した．

4.2.5 バックエンド：Rack サーバ

本実装では，バックエンドサーバとして，Rack を利用した[23]．Rack は Ruby 言語でアプリケーションを作るフレームワークである．MEC-RM はこのフレームワークを利用し，大まかに二つのサービスを実装した．

一つ目は，/data エンドポイントで，これはデータ保存サービスである．POST されたデータをハッシュ計算し Hashed Object に保存し ID を割り当てる．GET リクエストに対して，ID でハッシュを調べ，保存されたファイル名を SendFile ヘッダーに乗せ，nginx にこのファイルを送る旨を伝える．

二つ目は，/json-rpc エンドポイントで，これは MEC-RM で一番重要なコンポーネントである．ここでクライアントからのリクエストを処理し，他のコンポーネントにメッセージを送付する処理を行う．/json-rpc エンドポイントに送られたリクエストに対して，まずフォーマットチェックを行う．JSON-RPC ではないコンテンツには，エラーメッセージを直接返す．JSON-RPC のリクエストである確認ができたら，セッション情報を確認し，リクエスターのロールを判断する．もしセッション情報を特定できなかったら，要認証というエラーメッセージを返す．認証は JSON-RPC が提供した login メソッドを利用する．セッション情報はこのメソッドを実施するため，特例としてセッション情報がなくても呼び出されるようにした．

セッション情報については，JSON-RPC にセッション情報を持たせる仕組みとして実装した．HTTP セッションはホスト名に関連付けられ，ネットワーク状況変化により引き続きがうまくできないことがある．そのため，MEC-RM では利用せず，次の方法で実現するものとした．セッションが続いたリクエストでは，次の 4 つのメソッドを提供した．

1. job.fetch：ワーカーがジョブの割り当てを請求するメソッド．このメソッドを呼び出すワーカーは利用可能状態に設定し，ジョブがあれば割り当てる・
2. job.queue：リクエスターがジョブの実行を請求するメソッド．このメソッドからジョブの ID を振り当てて，ジョブ状態を確認する．
3. job.wait：リクエスターがジョブの実行状況を確認するメソッド．ジョブが終了したら，このメソッドでジョブの終了状態と出力データの ID を伝える．
4. job.update：ワーカーがジョブの実行状況を更新するメソッド．前のジョブが完了した状態で，本メソッドを呼び出した場合，job.wait メソッドを経由し，リクエスターに update を通知する．もし，前のジョブが失敗状態で本メソッドを呼び出した場合，ジョブのタグ（再実行制限）に従い，再実行や，失敗したことをリクエスターに伝える．

この 4 つメソッドでジョブの請求や割り当てを行う．

4.2.6 分散型 ID 生成アルゴリズム

ジョブやセッションを管理するため，ID が必要である．スケジューラはこの ID を参照してジョブの実行順序を決めるため，UUID など採番できない ID アルゴリズムは利用できない．また，ソートなどの操作を効率的に実現するためには，64bit に収まる ID が望ましい．こうした要求に対応するため Twitter 社は Snowflake ID を提案した．Snowflake ID の設計は図 5 で示した．Snowflake ID は一つの ID が 64bit に収まり，分散可能，かつ高速な採番が可能という目的で設計された ID 生成アルゴリズムである．先頭にタイムスタンプを持つため，順序があることが特徴とされる．本特徴により，TwitterID から Discord チャット ID まで広く使われている．

Snowflake Identifier

timestamp: 41bit. Milliseconds timestamp
host id: 10bit. Identifier for each machine.
seq: 12bit. Sequence number in same time.

図 5　SnowflakeID

しかし，この Snowflake ID を MEC-RM 環境で利用するには問題がある．それは，10bit のインスタンス ID では足りない，という問題である．MEC-RM はエッジコンピューティング環境を想定して，分散されているサーバの数を多く見積もっている．しかし 10bit のインスタンス ID では，マシンを 1024 台以上に拡張できない．このことから，Snowflack ID の拡張が必要である．一方，12bit のシーケンス ID はこのシステムの処理能力を超えている．エッジコンピューティング環境では,重い計算を処理することを想定しているため，実際にはミリ秒単位で処理できる作業が少ない．ミリ秒間 4096 回も実行できる計算には，ネットワーク遅延を加算して不効率になる．このため，シーケンス ID の部分を削減し，削減した分をインスタンス ID に譲ることが 可能である．

SakuraFubuki Identifier

timestamp: 40bit. Centiseconds (10 ms) timestamp
host id: 16bit. Identifier for each machine.
seq: 8bit. Sequence number in same time.

図 6　Sakura-Fubuki ID

このことから，シーケンス ID 領域を削減し，インスタンス ID に譲る考え方で，MEC-RM 用分散型 ID 生成アルゴリズムを定義した．このアルゴリズムには，40bit のタイムス

タンプを利用し，最初にこのアルゴリズムを実装した年の元日（2019 年 1 月 1 日）からのセンチ秒（10 ミリ秒）を載せるものとした．インスタンス ID は 16bit まで拡張した．拡張の代償としてシーケンス ID の部分を 8bit まで削減した．この改造したアルゴリズムは最大 65536 インスタンスから毎秒 25600 この ID を生成できる．このように，多くのインスタンスから ID を生成でき，かつ分散型で生成することから， Sakura-Fubuki ID と名付けた．

4.2.7 タグに基づいた多 FIFO スケジューラ

ジョブの実行を支援するため，Tag-Based Multi-FIFO ポリシーを実施するスケジューラを実装した（図 7）．

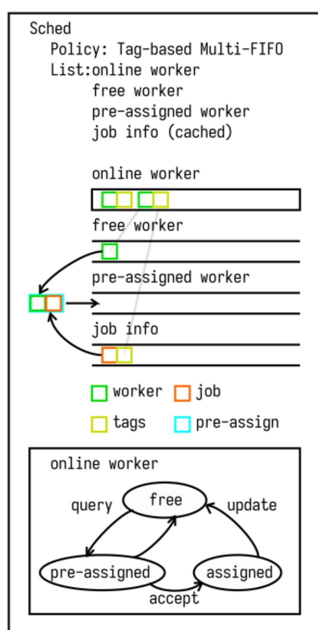

図 7　Tag-Based Multi-FIFO スケジューラ

ジョブとワーカーは各自タグ情報を持つ．ワーカーは実行可能なジョブなど，実行能力に関わるタグがつけられてる．ジョブ名自身はタグになり，ジョブを請求した時には他のタグ（緊急ではないなど）も追加できる．スケジューラはジョブキュー，利用可能なワーカーの二つのキューから，それぞれ一つずつ取り出し，マッチング操作を行う．マッチング操作では，実行能力タグを参照し，このマッチングを優先的に行う．実行能力タグのマッチングができたら，さらに緊急状態などのタグをマッチングする．もし全てのタグをマッチングできたら，そのワーカーにジョブを割り当てる．一部のタグしかマッチングできない場合は，実行能力タグが全部マッチングしたワーカーの中で，マッチングできたタグが一番多かったワーカーのジョブを割り当てる．本設計により，ジョブを常に実行可能なワーカーに割り当てることができる．

5. 評価

本提案システムの有効性を確認するため，次の評価を行った．

5.1 排他利用および実行評価

まず始めに，本研究で示した課題であるマルチ FPGA ボードにおける排他利用の実現を確認するため，計算資源利用者としてマネージャーにジョブを送信した場合の時間を計測した．三つのプロセスから，独立してジョブを送信する．約 1 秒ごと一個のジョブを送信する頻度で，各プロセスから 200 個のジョブを送信した．送ったジョブは一つのワーカーで約 1 秒程度の実行時間で実施された．実験するときはワーカーを計 4 つ接続した．

5.2 実験環境

実験は以下のような環境で実施した．マネージャーは Raspberry Pi 4 で実行している．8GB メモリを保有した型番を利用した．システムは Ubuntu 20.04 LTS server である．ワーカーとして，3 節に紹介した M-KUBOS ボードと GPU が乗せた nVIDIA Jetson ボードを各二枚用いた．x86 マシンを一台，リクエスターとしてジョブを発行した．すべてのマシンは有線ネットワークにつないで，有線ネットワークは 1000Base-T を利用した．

5.3 実験結果

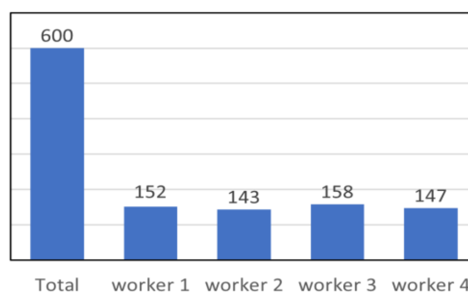

図 8 各ワーカーに割り当てられたジョブ数

排他利用についての実験の結果を図 8 で示した．

図 8 の結果から，各ワーカーがほぼ同数（平均 150）のジョブを実行し，すべてのワーカーが実行したジョブの数は請求したすべてのジョブの数と同一であることがわかった．このことから，後に到着したジョブは前のジョブとの競合が発生していないことがわかる．本結果で，一つ目の課題である排他制御の課題を解決したことを確認できた．

ⓒ2022 Information Processing Society of Japan

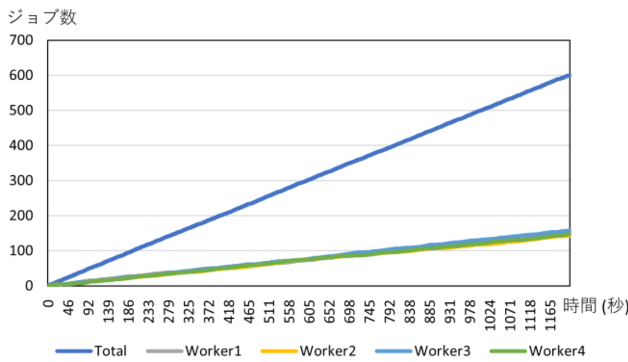

図 9 ジョブ割り当ての時間推移

図 9 は実行時のジョブ割り当て状況を示した．発行されたジョブ数（青い線）の増加により，各ワーカーが実行したジョブの数も増加していることが理解できる．また，各ワーカーが実行したジョブの数は，時間ごとに大きな差が出なかった．これは，発行されジョブが，大体各ワーカーに分散されたためと考えられるためである．また，ジョブのスケジューリングが行われ，ジョブが異なるワーカーでも実行されていたことが確認できた．本結果で，二つ目の課題を解決できたことを確認できた．

5.4 障害対応の評価

本提案システムが，ワーカーであるアクセラレータの故障に柔軟な対応ができるかについては模擬的に障害を発生させる評価を行った．具体的には，計算資源利用者は，ジョブをリクエストし，ワーカーがジョブをスケジューリングする際に，一台のワーカーのケーブルを抜き，オフラインにする．そして一定時間後，またケーブルを挿し，オンラインに戻す．このように，ワーカーに故障が発生した状況を模擬した評価を行った．結果を図 10 で示した．

図 10 では，黄色いワーカー（Worker4）が，故障を模擬し

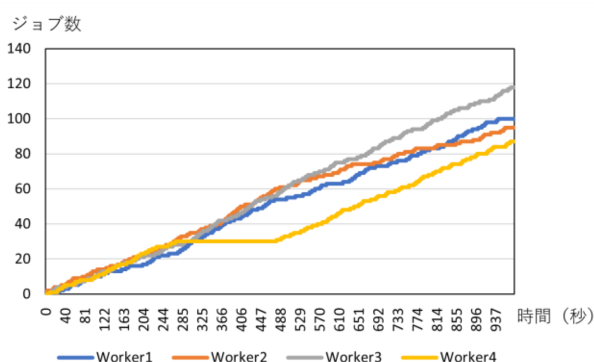

図 10 故障が発生した時の対応

た一台のワーカーである．それ以外（Worker1-3）は，故障していないものとする．Worker4 に対して，270 秒前後（時刻 1）にケーブルを抜き，480 秒（時刻 2）でケーブルを挿しなおして，実行した結果を示している．270 秒前後（時刻 1) でケーブルを抜いた後，故障ワーカー（黄色い線，Worker4）は，ジョブの数の変化がなくなった．理由として，MEC-RM マネージャーがこのワーカーへのジョブ割り当てを，いったん中止したためである．Worker4 以外の Worker1–3 では，微量程度に増加率が高くなり，継続している．理由として，Worker4 に割り当てた分が残った三つのワーカーに割り当てられたためである．

480 秒（時刻 2）にケーブル挿しなおし，故障の回復を模擬した．図 10 で見ると，Worker4 の増加もこの時から回復し，残り三本の線の増加率も低くなった．それはマネージャーが故障したワーカーの回復をすぐ検知でき，中止したジョブの割り当ても再開したためである．この結果でこのシステムがワーカーの障害を柔軟に対応できることが確認でき，三つ目の課題を解決できたことを確認した．

6. 結論

本研究はヘテロジニアス環境向けたリソースマネージシステムを実装し，評価を行った．実装したリソースマネージャーは異種混在の計算環境に対応でき，計算資源の割り当てができた．さらに故障が発生した時でも対応できる．今後の研究について，以下の方向で検討している．1). JSON-RPC の通信負荷に着目し，バイナリフォーマット（message pack）などで通信負荷を減らす．2). 各アクセラレータの特性を考え，ジョブの内容によってアクセラレータを自動選択するアルゴリズムの実装などを行う予定である．

謝辞 本研究は JST, CREST, JPMJCR19K1 の支援を受けたものです．ここに感謝の意を表します．

参考文献

[1] 内閣府: Society 5.0，https://www8.cao.go.jp/cstp/society5_0/
[2] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang and W. Zhao, "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications," in IEEE Internet of Things Journal, vol. 4, no. 5, pp. 1125-1142, Oct. 2017, doi: 10.1109/JIOT.2017.2683200.
[3] M.D. Donno, K. Tange, and N. Dragoni, “Foundations and evolution of modern computing paradigms: Cloud, IoT, edge, and fog,” IEEE Access, vol.7, pp.150936–150948, Oct. 2019.
[4] J. Yuan and X. Li, “A reliable and lightweight trust computing mechanism for IoT edge devices based on multi-source feedback information fusion,” IEEE Access, vol.6, pp.23626–23638, April 2018.
[5] 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, vol.130, pp.94–120, Jan. 2018.
[6] M. Chiang and T. Zhang, "Fog and IoT: An Overview of Research Opportunities," in IEEE Internet of Things Journal, vol. 3, no. 6, pp. 854-864, Dec. 2016, doi: 10.1109/JIOT.2016.2584538.
[7] Koki Nagahama, Yoichi Ishiwata, Midori Sugaya, “Resource utilization prediction model for SLAM offload to Edge”, Proceedings of IEEE CANDAR (The Eighth International Symposium on Computing and Networking). 2020.
[8] Kyoung-Su Oh, Keechul Jung, GPU implementation of neural networks, Pattern Recognition 37, 6, 1311-1314, 2004
[9] Sannomiya Natsuho, Takeshi Ohkawa, Hideharu Amano, Midori Sugaya, “Power Consumption Reduction Method and Edge

Offload Server for Multiple Robots”, Edge 2022, Pages 1-19. Conference proceedings EDGE 2021.
[10] 工藤 知宏, 高野 了成, 天野 英晴, 鯉渕 道紘, 松谷 宏紀, 塙 敏博, 池上 努, 須崎 有康, 田中 哲, 赤沼 領大, 並木 周, 田浦 健次朗：“データの流れに着目した異種エンジン統合クラウドシステム Flow in Cloud”, 信学技報 vol. 117 no. 153 CPSY2017-16, pp. 1-5 (2017).
[11] 山倉 美穂, 高野 了成, Akram Ben Ahmed, 天野 英晴：“マルチ FPGA クラウドシステムにおけるマルチテナント式資源管理の開発”, 信学技報, vol. 120, no. 168, RECONF2020-21, pp. 13-18, (2020).
[12] B. Cope, P. Y. K. Cheung, W. Luk and L. Howes, "Performance Comparison of Graphics Processors to Reconfigurable Logic: A Case Study," in IEEE Transactions on Computers, vol. 59, no. 4, pp. 433-448, April 2010, doi: 10.1109/TC.2009.179.
[13] Q. Wu, Y. Ha, A. Kumar, S. Luo, A. Li and S. Mohamed, "A heterogeneous platform with GPU and FPGA for power efficient high performance computing," 2014 International Symposium on Integrated Circuits (ISIC), 2014, pp. 220-223, doi: 10.1109/ISICIR.2014.7029447.
[14] N. Hu, X. Zhou, X. Li and C. Wang, "3D Waveform Oscilloscope Implemented on Coupled FPGA-GPU Embedded System," 2018 5th International Conference on Information Science and Control Engineering (ICISCE), 2018, pp. 1-5, doi: 10.1109/ICISCE.2018.00010.
[15] A. Khawaja, J. Landgraf, R. Prakash, M. Wei, E. Schkufza and C. J. Rossbach: “Sharing, protection, and compatibility for reconfigurable fabric with amorphos”, 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), Carlsbad, CA, USENIX Association, pp. 107–127 (2018).
[16] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone and J. C. Phillips, "GPU Computing," in Proceedings of the IEEE, vol. 96, no. 5, pp. 879-899, May 2008, doi: 10.1109/JPROC.2008.917757.
[17] E. Lindholm, J. Nickolls, S. Oberman and J. Montrym, "NVIDIA Tesla: A Unified Graphics and Computing Architecture," in IEEE Micro, vol. 28, no. 2, pp. 39-55, March-April 2008, doi: 10.1109/MM.2008.31.
[18] C. Xiong and N. Xu, "Performance Comparison of BLAS on CPU, GPU and FPGA," 2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), 2020, pp. 193-197, doi: 10.1109/ITAIC49862.2020.9338793.
[19] Torch http://torch.ch/
[20] TensorFlow https://www.tensorflow.org/
[21] Nginx, https://nginx.org/en/
[22] Redis, https://redis.io/
[23] Rack, https://github.com/rack/rack

ⓒ2022 Information Processing Society of Japan