コンテナ型クラウドサービス基盤におけるハードウェア仮想化技術の併用によるネットワーク制御の検討

中田 裕貴[1,a)]　松原 克弥[1,b)]　松本 亮介[2,c)]

概要：マルチテナント方式のクラウドサービス基盤では，各テナントが平等にリソースを使用できる公平性，テナント間分離による性能低下を抑えたインスタンス性能，既存アプリケーションを改変せずに実行可能とする利便性の 3 つのサービス品質に十分な配慮を行う必要がある．近年注目されているコンテナ型仮想化技術によるクラウドサービス基盤では，複数のテナントそれぞれに対して，1 つのプラットフォーム上で動作するプロセス単位で分離したインスタンスを提供する．その際に広く用いられている OS によるネットワークインタフェースの多重化方式は，パケット送受信の際に多くの OS 機能を経由することでネットワーク I/O 性能が低下する課題がある．この課題に対して，OS カーネルのバイパスなどの高速にパケット処理を行うパケット送受信経路の最適化技術や，軽量な OS 機能の実装などの OS 処理に対する最適化技術がある．しかし，これらの手法は，トラフィックの公平性，ネットワーク I/O 性能，利便性のサービス品質すべてを最適にすることが難しい．本研究では，各テナントに割り当てたプロセスのそれぞれに対して，ハードウェア仮想化技術を用いた軽量ハイパーバイザにより実現する仮想ネットワークインタフェースを提供し，ネットワーク I/O 性能と利便性を最大限維持しつつ，それら仮想ネットワークインタフェースへのトラフィック制御によりテナント間の公平性を実現する手法を提案する．本手法は，コンテナ型仮想化技術向けのネットワーク機能においてハードウェア仮想化技術を併用することで，OS カーネルのバイパスなどを使用した際のトラフィック制御，複数の OS 機能間経由にともなう冗長処理削減による性能低下抑制，既存 OS 向けアプリケーションをそのまま利用できる利便性を両立できる．

1. はじめに

クラウドサービス事業者が，利用者（以下，テナント）にアプリケーション実行環境（以下，インスタンス）を提供する方式として，単一サーバ上で複数のテナントに対してインスタンスを提供するマルチテナント方式がある [1]．マルチテナント方式のクラウドサービスでは，単一サーバ上で高集積にインスタンスを配置することによるハードウェアコストの低減を実現するために，Docker[2] などの OS プロセス単位でインスタンスを作成するコンテナ型仮想化技術の使用が増加している [3][4][5]．コンテナ型仮想化技術を用いたクラウドサービスを実現する際，サーバリソースの管理や分離のために，サーバ上に複数の仮想マシンを作成し，各テナント毎に仮想マシンを使用してクラウドサービスを提供することがある．しかし著者らは，さらなる高集積のために，各インスタンスが単一 OS 上で提供されるコンテナ型クラウドサービスの実現を目指している．

マルチテナント方式のクラウドサービスでは，インスタンス間のリソース分離が不十分な場合，各テナントの公平性を実現できないことがある．例えば，あるテナントのインスタンスがサーバ上のリソースを過剰に使用し，他のテナントのインスタンスは同一の使用料金でありながら，通常より少ないリソースしか使用できずに不平等な性能になることが想定される．そこで，クラウドサービス事業者には，インスタンス間の分離とリソース使用の公平性を担保し，分離によるインスタンスの性能低下を最小限に抑えることが求められる [6][7][8]．また，既存のクラウドサービスでは，多くのテナントで広く用いられているアプリケーションの多くが，コードを変更することなく動作する Linux[9] をインスタンス内で利用できる OS 環境として提供することで，利便性を確保している [10]．以上で述べたように，クラウドサービス事業者は，リソース使用の公平性担保，インスタンスの性能，インスタンスの利便性の 3 要件を考慮する必要がある．

コンテナ型仮想化技術を用いたマルチテナント方式のク

[1] 公立はこだて未来大学
Future University Hakodate
[2] さくらインターネット株式会社 さくらインターネット研究所
SAKURA internet Research Center, SAKURA internet Inc.
[a)] g2120032@fun.ac.jp
[b)] matsu@fun.ac.jp
[c)] r-matsumoto@sakura.ad.jp

ラウドサービスでは，各テナントはプロセス単位の分離を意識せず，インスタンス間での IP アドレスベースの通信や，各テナントが同一ポート番号の使用を前提としたアプリケーションの実行を行うことがあるため，インスタンスの利便性という要件を満たすために，ユーザ環境の分離以外にもネットワークに関連する機能の分離・制御が重要である．そのため，テナントのアプリケーションがコンテナ型のインスタンスでも変更することなく動作するために，各インスタンスへの IP アドレスの付与や，独立したネットワークインタフェースの付与などのネットワークに関連する機能の分離をする必要がある．また，各インスタンスが平等にネットワーク帯域を使用できる，ネットワークリソースに対する公平性も担保する必要がある．さらに，以上の 3 要件を満たしながら，コンテナ型仮想化技術の特徴である，単一サーバ上でのインスタンス高集積化のメリットを最大限維持することが重要となる．

コンテナ型仮想化技術を用いて作成したインスタンス毎にネットワーク機能を分離・制御する既存手法として，OS のブリッジ機能とパケットフィルタリング機能の利用した手法，専用の制御コードを OS カーネル内で実行する手法，OS カーネルをバイパスする手法，I/O 仮想化支援ハードウェアを用いた手法，軽量な OS 機能の実装手法がある．しかし，これらの手法は，マルチテナント方式のクラウドサービスで必要な 3 要件全てを満たすことや，コンテナ型仮想化技術の特徴である高集積の維持に課題がある．

本研究では，コンテナ型仮想化技術を用いて作成したインスタンスに対して，ハードウェア仮想化技術を用いた軽量ハイパーバイザによって仮想ネットワークインタフェースを提供し，ネットワークトラフィックを制御する手法を提案する．仮想ネットワークインタフェースに対してハイパーバイザからネットワークトラフィック制御することで，各インスタンスが動作するサーバと別サーバや外部との通信におけるネットワークトラフィック制御を実現し，リソース使用の公平性担保の要件を満たせる．また，仮想ネットワークインタフェースを軽量ハイパーバイザ内で作成し，それらをプロセス単位で分離されたインスタンスが使用することで，OS のブリッジ機能とパケットフィルタリング機能の利用した手法と比べ，OS カーネル内のネットワークスタックを経由する回数の削減を目指す．これにより，ネットワーク I/O 性能を最大限維持することでインスタンスの性能の要件を満たす．さらに，軽量ハイパーバイザ上でネットワーク機能の分離・制御を行うことで，テナントに対して既存アプリケーションの移植を要求せず，インスタンスの利便性の要件を満たすことができる．

本稿の構成について述べる．2 章では，マルチテナント方式のクラウドサービスにおけるリソース使用の公平性，インスタンスの性能，インスタンスの利便性の 3 要件と，インスタンス集積度の観点から，コンテナ型仮想化技術におけるネットワーク分離・制御の既存手法を整理し，それらの課題についてまとめる．3 章では，ハードウェア仮想化技術の併用によるネットワーク機能の分離・制御手法について提案する．4 章では，本提案手法の実装方針について論じる．5 章では，提案手法の基本機能であるネットワークトラフィック制御機能に関する予備実験を行い，その有効性を確認する．最後に，6 章において，提案内容についてまとめ，今後の課題を示す．

2. コンテナ型仮想化技術における ネットワーク分離・制御の既存手法と課題

コンテナ型仮想化技術を用いたインスタンス毎にネットワーク機能を分離・制御を行う既存手法として，以下の 5 つの手法がある．

(1) OS のブリッジ機能とパケットフィルタリング機能の利用
(2) 専用の制御コードを OS カーネル内で実行
(3) OS カーネルのバイパス
(4) I/O 仮想化支援ハードウェアの使用
(5) 軽量な OS 機能の実装

コンテナ型仮想化技術を用いたマルチテナント方式のクラウドサービスにおけるネットワーク分離では，1 章で述べたように，ネットワークトラフィックの公平性担保，ネットワーク I/O 性能，インスタンスの利便性の 3 要件を考慮しつつ，コンテナ型仮想化技術の特徴である高集積なインスタンス配置を最大限維持することが重要である．本章では，コンテナ型仮想化技術におけるネットワーク分離・制御の既存手法をこれらの観点で整理し，それらの課題について述べる．

2.1 OS のブリッジ機能と パケットフィルタリング機能の利用

OS が提供する仮想ネットワークインタフェース（veth）を各インスタンスに対して提供し，それらのネットワークインタフェースを介する通信を OS のブリッジ機能とパケットフィルタ機能で制御する手法（図 1）がある [11][12]．この手法は，OS が提供する仮想ネットワークインタフェースに対して OS のネットワークトラフィック制御機能を使用し，インスタンス毎に使用可能なネットワーク帯域を指定することで，OS 機能でネットワークトラフィックの公平性を担保できる [13]．また，インスタンスに対して仮想ネットワークインタフェースを提供することで，インスタンス上での動作のために既存のアプリケーションを改変する必要がなく，インスタンスの利便性を保っている．さらに，OS 機能でネットワーク分離・制御を行うことで，コンテナ型仮想化技術の特徴である高集積なインスタンス配置を最大限維持できる．しかし，パケット送受信時に OS カーネルのネットワークスタックを経由する回数が増加

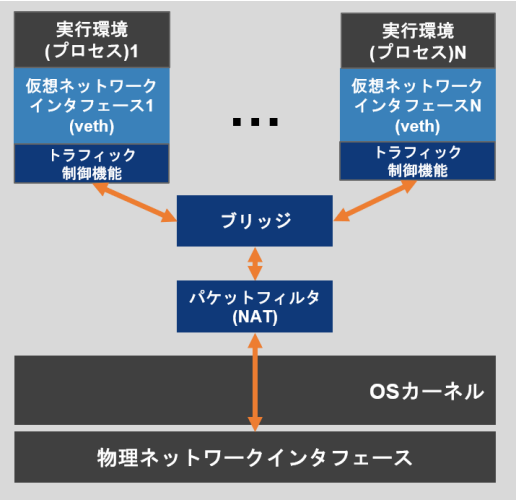

図 1　OS のブリッジ機能とパケットフィルタリング機能によるネットワーク分離

することで，ネットワーク I/O 性能が低下する課題がある [14].

この手法では，ネットワークトラフィックの公平性とインスタンスの利便性の要件は満たし，高集積なインスタンス配置を最大限維持できているが，ネットワーク I/O 性能は十分に満たせていない.

2.2　専用の制御コードを OS カーネル内で実行

OS が提供する仮想ネットワークインタフェース（veth）に対する通信を，OS カーネル内で実行する専用のバイトコードで実装したプログラムで制御する手法（図 2）がある [15]. この手法は，2.1 節で使用している OS のブリッジ機能とパケットフィルタリング機能を eBPF（extended Berkeley Packet Filter）[16] と呼ばれるカーネル内制御プログラムで置き換えることで，OS のネットワークスタックを経由する回数を削減し，単一サーバ内のインスタンス間通信におけるネットワーク I/O 性能を向上している [17]. また，OS カーネル内で分離・制御することで，2.1 節の手法と同様に分離によるリソース使用を抑え，単一サーバ上での高集積なインスタンス配置を最大限維持できる．さらに，eBPF で実装されたネットワークトラフィック制御機能を用いることで，OS カーネル内でネットワークトラフィックの公平性を担保できる．しかし，この手法でトラフィック制御する際に使用する設定ファイルでは，単一マシン内コンテナ間通信と別サーバや外部との通信を区別しない．そのため，ネットワークトラフィック制御機能を用いて各インスタンスに指定する使用可能な帯域は，サーバの物理ネットワークインタフェースで使用できる帯域と異なる．したがって，各インスタンスが動作するサーバと，別サーバや外部との通信におけるネットワークトラフィックの公平性担保に適さない.

この手法では，ネットワーク I/O 性能とインスタンスの利便性の要件は満たし，高集積なインスタンス配置を最大限維持できているが，ネットワークトラフィックの公平性担保は十分に満たせていない.

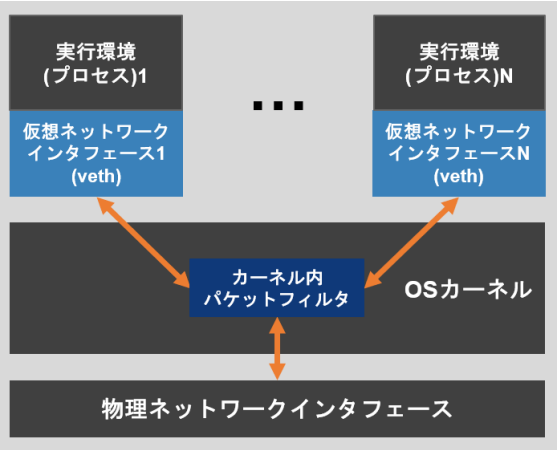

図 2　カーネル内制御コードを用いたネットワーク分離

上記の手法と同様に，Open vSwitch[18] は，2.1 節のブリッジとして使用する際に XDP（eXpress Data Path）と呼ばれるカーネル内制御プログラムを使用し，ネットワーク機能を分離・制御できるモードがある [19]. XDP を使用したモードの Open vSwitch をブリッジとして用いることで，OS カーネルのネットワークスタックを経由する回数を削減し，2.1 節の手法に比べ，ネットワーク I/O 性能が向上している．また，他の手法と同様に，インスタンス内で動作するアプリケーションを改変する必要がなく，インスタンスの利便性を保つことができる．さらに，OS カーネル内で分離や制御を行うことで，分離によるリソース使用を抑え，単一サーバ上での高集積なインスタンス配置を最大限維持できる．しかし，Open vSwitch では，XDP を使用したモードの場合はネットワークトラフィック制御機能が使用できず，トラフィックの公平性を担保できない [20].

この手法では，ネットワーク I/O 性能とインスタンスの利便性の要件を満たし，高集積なインスタンス配置を維持できているが，ネットワークトラフィックの公平性担保は満たせていない.

2.3　OS カーネルのバイパス

Open vSwitch[18] は，2.1 節のブリッジとして使用する際，OS カーネルをバイパスする技術である DPDK（Data Plane Development Kit）[21] を用いてパケット送受信を行うモードがある [22]. DPDK を使用したモードの Open vSwitch をブリッジとして使用し，OS カーネルをバイパスしてパケット送受信することで，2.1 節の手法に比べ，ネットワーク I/O 性能が向上している．また，Open vSwitch のネットワークトラフィック制御機能を使用することで，ネットワークの公平性を担保することができる．さらに，他の手法と同様に，インスタンス内で動作するアプリケー

ションを改変する必要がなく，インスタンスの利便性を保つことができる．

この手法では，ネットワークトラフィックの公平性担保とネットワーク I/O 性能，インスタンスの利便性の 3 つの要件全てを満たせている．しかしながら，DPDK は，ネットワーク I/O 性能向上を実現するために，特定の CPU コアを専有してパケット送受信処理を行う．そのため，各インスタンスが使用可能な CPU リソースが減少し，単一サーバ上で実行できるインスタンスの集積度が低下してしまう課題がある．

上記の手法で使用した DPDK に加え，FreeFlow[23] は，RDMA（Remote Direct Memory Access）と呼ばれるインスタンス間で OS カーネルをバイパスしたメモリ共有による通信によって，ネットワーク分離・制御を行う．この手法では，単一サーバ内のインスタンス間通信は RDMA を使用してメモリを共有し，別サーバとの通信は DPDK を用いてパケットの送受信に置き換えることで，ネットワーク I/O 性能を向上している．また，他の手法と同様に，インスタンス内で動作するアプリケーションを改変する必要がなく，インスタンスの利便性を保つことができる．しかし，FreeFlow はカーネルをバイパスする DPDK や RDMA を使用することで，OS のトラフィック制御機能が使用できず，トラフィックの公平性を担保できない．

この手法では，ネットワーク I/O 性能とインスタンスの利便性の要件を満たせているが，ネットワークトラフィックの公平性は満たせていない．また，上記の手法と同様に，DPDK を用いることで，各インスタンスが使用可能な CPU リソースが減少し，単一サーバ上で実行できるインスタンスの集積度が低下してしまう．

2.4 I/O 仮想化支援ハードウェアの使用

各インスタンスに対して，SR-IOV（Single Root I/O Virtualization）と呼ばれる I/O 仮想化支援ハードウェアの物理ネットワークインタフェース（PF）自身が仮想ネットワークインタフェース（VF）を作成する機能を用いて，ネットワーク機能を分離・制御する手法（図 3）がある [24]．この手法では，仮想ネットワークインタフェース（VF）に対して，使用可能な帯域を指定することで，ネットワークトラフィックの公平性を担保できる [25]．また，OS の機能ではなく，物理ネットワークインタフェース（PF）の機能を用いて仮想ネットワークインタフェース（VF）を提供することで，OS カーネルの経由回数を削減し，ネットワーク I/O 性能を向上している [14]．さらに，他の手法と同様に，OS 上のインスタンスで動作するアプリケーションを改変する必要がなく，インスタンスの利便性を保つことができる．

この手法では，ネットワークトラフィックの公平性担保とネットワーク I/O 性能，インスタンスの利便性の要件を満たせている．しかしながら，SR-IOV で割り当て可能な仮想ネットワークインタフェース（VF）数は 256 が上限であり [26]，実際はネットワークインタフェースの実装次第で更に少ない場合がある．そのため，SR-IOV をコンテナ型仮想化技術を用いたマルチテナント方式のクラウドサービスに用いた場合，単一サーバ上の集積度が低下してしまう課題がある．

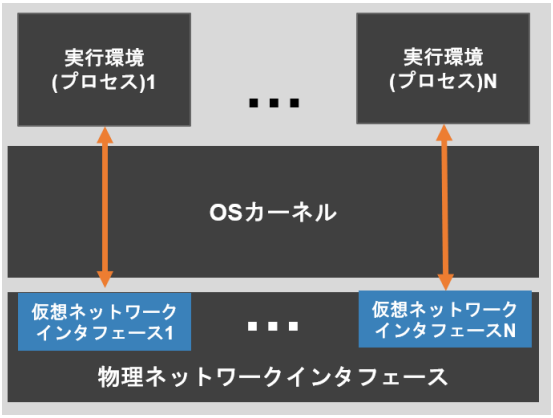

図 3　I/O 仮想化支援ハードウェアの使用によるネットワーク分離

2.5 軽量な OS 機能の実装

クラウド専用の OS を実装して，その OS 機能内でネットワーク分離・制御を実現する手法がある [27]．この手法では，インスタンス間のネットワーク分離を行うために専用 OS を実装し，専用 OS 上のパケット送受信機能においてネットワーク I/O 性能を向上するために DPDK を使用している．また，各インスタンスからパケット送受信機能に対してデータコピーを行う際に，ネットワークトラフィック制御を行うことで，OS カーネルをバイパスする手法を使用した場合でも，OS の機能でネットワークトラフィックの公平性を担保している．さらに，独自の軽量プロセスでインスタンスを提供することで，単一サーバ上での高集積なインスタンス配置を維持できている．しかし，独自 OS を用いた手法は，OS 上のインスタンスで動作するアプリケーションの移植をテナントに強制する必要があり，クラウドサービスを利用する際の利便性が低下する．

この手法では，ネットワークトラフィックの公平性担保とネットワーク I/O 性能の要件を満たせているが，インスタンスの利便性は満たせていない．

OS のソケットを拡張し，OS のブリッジやパケットフィルタリング機能を経由せずに，ネットワークを分離・制御する手法がある [28]．この手法では，仮想ネットワークインタフェースにつながるコンテナ内通信ソケットの入出力を物理ネットワークインタフェースにつながるコンテナ外の通信ソケットへフォワードすることで、送受信における各ネットワークインタフェースに関わるネットワークスタック処理の重複を除去することで，ネットワーク I/O 性能を

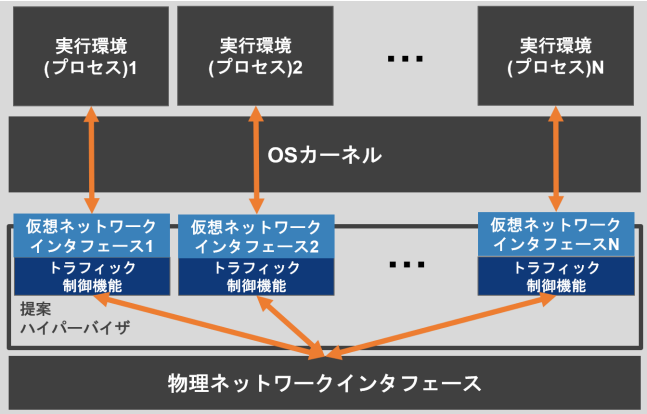

図 4　提案手法のアーキテクチャ

向上させている．また，ソケットを拡張することで，分離や制御によるリソース使用を抑え，高集積なインスタンス配置を最大限維持できる．しかし，アプリケーションの使用するソケットを提案手法に置き換えるために，ソケットに関連するシステムコールをフックしているため，静的なライブラリのリンクや特定のプログラミング言語を用いたアプリケーションでは置き換えることができない．そのため，拡張ソケットをインスタンス上のアプリケーションで使用するには，アプリケーションを改変する必要があり，インスタンスの利便性が低下する．さらに，OS のソケットレイヤ以降をバイパスすることで，OS のネットワークトラフィック制御機能を用いてトラフィック制御できず，トラフィックの公平性を担保できない．

この手法では，ネットワーク I/O 性能の要件を満たし，高集積なインスタンス配置を維持できているが，ネットワークトラフィックの公平性担保とインスタンスの利便性は満たせていない．

3. ハードウェア仮想化技術を用いたコンテナ向けネットワーク分離・制御手法の提案

本研究では，ネットワークトラフィックの公平性担保，ネットワーク I/O 性能，インスタンスの利便性の 3 要件を考慮しつつ，コンテナ型仮想化技術の特徴である高集積を最大限維持するために，ハードウェア仮想化技術を用いて作成した仮想ネットワークインタフェースを，コンテナ型仮想化技術を用いたインスタンスに提供する手法を提案する．本提案では，図 4 のように，軽量ハイパーバイザで作成した仮想ネットワークインタフェースを各インスタンスに提供することで，インスタンス毎のネットワーク分離を実現する．

本提案では軽量ハイパーバイザ内で，仮想ネットワークインタフェースに対してネットワークトラフィック制御を行う．このネットワークトラフィック制御機能により，各インスタンスが動作するサーバと別サーバや外部との通信におけるネットワークトラフィック制御を実現し，ネットワークトラフィックの公平性の要件を満たすことができる．さらに，OS より下の層でネットワークトラフィック制御を実現することで，OS カーネルをバイパスする DPDK などの高速パケット処理技術を用いた場合でも，ネットワークトラフィックの公平性を担保できる．

また，ネットワーク分離・制御を OS ではなく，軽量なハイパーバイザを用いて実現することで，OS のネットワークスタックを経由する回数を削減する．これにより，S のネットワークスタックを経由する回数を起因とするボトルネックを削減し，ネットワーク I/O 性能を最大限維持することで，ネットワーク I/O 性能の要件を満たすことを目指す．

さらに，OS より下の層で動作する軽量ハイパーバイザでネットワーク分離を行う際，クラウド用の専用 OS や OS 上に専用の機能を用いない．そのため，様々な OS 環境で開発されたテナントのアプリケーションが，クラウドサービス上のインスタンスでも変更することなく動作可能であるため，利便性の要件を満たすことができる．

コンテナ型仮想化技術の特徴である高集積なインスタンス配置を最大限維持するために，本提案手法の軽量ハイパーバイザは，ネットワーク機能の分離と制御のみを行う．これにより，ハイパーバイザが使用する CPU やメモリなどのリソース消費を削減し，インスタンスが利用するリソースへの影響を最小限にする．また，I/O 仮想化支援ハードウェアではなく，ハイパーバイザを用いて仮想ネットワークインタフェースを作成することで，仮想ネットワークインタフェースの個数が物理ネットワークインタフェースの実装に依存しない．

4. ハードウェア仮想化技術を用いたコンテナ向けネットワーク分離・制御手法の実装

本提案手法は，各インスタンスに対してネットワークインタフェースを提供する仮想ネットワークインタフェース作成機能と，各インスタンスのネットワークリソース使用を公平性を保つために使用するネットワークトラフィック制御機能の 2 つによって実現する．また，通信には，単一マシン内のインスタンス間通信と，別マシン上のインスタンスや外部との通信の 2 種類が存在するが，本稿では，別マシン上のインスタンスや外部との通信が対象の実装手法について述べる．

4.1 仮想ネットワークインタフェース作成機能

仮想ネットワークインタフェース作成機能を実現する際，ネットワークデバイスのみを仮想化するという技術的課題がある．本提案手法では，インスタンス間のネットワーク分離・制御のみ行う軽量ハイパーバイザを実現することで，コンテナ型仮想化の特徴である，単一サーバ上での高集積なインスタンス配置を最大限維持する．そのため，ス

トレージや USB デバイスなどのネットワーク分離に不要なデバイスは仮想化せず，ネットワークデバイスのみ選択して仮想化することが必要である．

ネットワークデバイスのみを選択して仮想化するために，準パススルーハイパーバイザの BitVisor[29] を用いる．BitVisor はセキュリティ向上が目的のハイパーバイザであり，ネットワークデバイスなどの I/O を監視し，暗号化やアクセス制御を実現できる．BitVisor が採用している準パススルー型アーキテクチャは，全てのデバイスを仮想化せず，仮想化が必要なデバイスのみを選択して仮想化できる．BitVisor を用いることで，ネットワークデバイスのみの仮想化という技術的課題を解決できる．また，BitVisor のメモリフットプリントは 64MB から 128MB 程度と小さく，OS が使用するメモリ領域を圧迫しないことで，単一サーバ上での高集積なインスタンス配置を最大限維持できる．

BitVisor でネットワークインタフェースを仮想化する際，virtio-net と呼ばれるハイパーバイザと OS が連携することを前提とした方式を，準パススルードライバとして用いることができる．virtio-net では，ハイパーバイザと OS が連携することで，送受信処理の簡略化や単純化を実現している，しかし，BitVisor に実装されている virtio-net などの準パススルードライバでは，OS から物理ネットワークインタフェースへの制御を捕捉してネットワーク I/O 内容の把握や加工を行うために，実インタフェースと仮想化されたインタフェース間で 1 対 1 の対応を持つように設計されている．そのため，準パススルードライバでは，1 対 1 の実装しか存在せず，本提案手法のように物理ネットワークインタフェースに対して複数の仮想ネットワークインタフェースを用意するような，1 対多の関係で仮想化することは考慮されていない．そこで，本提案手法では，仮想ネットワークインタフェースの作成に，対応する物理ネットワークインタフェースが存在しない virtio-net デバイスを作成可能な，BitVisor の仮想 virtio-net ドライバを拡張してネットワークインタフェースを多重化する．仮想 virtio-net は，BitVisor と OS 間でのログ出力などに用いることができる．本実装では，仮想 virtio-net をベースに使用されていない PCI デバイスのバス番号を用いて仮想ネットワークインタフェースを多重化し，I/O ポートのアクセスを検出した際に準パススルー virtio-net ドライバの送信処理を流用して送受信することで，複数の仮想ネットワークインタフェースを OS 上で動作する各インスタンスに提供する．

4.2 ネットワークトラフィック制御機能

BitVisor では，OS から物理ネットワークインタフェースへの制御を捕捉してネットワーク I/O 内容の把握や加工を行うためにを実現するために，シャドウ・バッファと呼ばれる複製したリングバッファを用いて，OS とネットワークインタフェース間でパケット送受信を行う．リングバッファは，ディスクリプタと呼ばれるパケットが格納された領域へのポインタなどが格納されている．BitVisor は，シャドウ・バッファを OS に対して見せることで，OS が使用するリングバッファと物理ネットワークインタフェースが使用するリングバッファを分離する．BitVisor はこの 2 種類のリングバッファの同期を適切なタイミングで行うことで，ネットワーク I/O 内容の把握や加工を実現している．

本提案手法のネットワークトラフィック制御機能は，シャドウ・バッファとバッファの同期処理の際にパケットサイズを取得し，1 秒あたりにコピーされたパケットの量を測定する．そして，シャドウバッファとバッファ間でデータコピーを行う際，予め設定した 1 秒あたりに送信できるパケットの上限値と比較し，その上限値を超えた場合は処理を待つことで制御を実現する．本稿では，ネットワークトラフィックの公平性を担保することができるか確認するために，BitVisor の準パススルードライバに対して，提案手法のネットワークトラフィック制御機能を実装した．

表 1 実験に用いたクライアントの仕様

OS	CentOS 7
Linux カーネル	5.4
CPU	Intel（R） Core（TM） i5-6260U CPU 1.80GHz
RAM	8GByte
NIC	Intel Coporation Ethernet Connection I219-V （rev 21）

表 2 実験に用いたサーバの仕様

OS	Ubuntu 18.04LTS
Linux カーネル	4.15
CPU	Intel（R） Atom（TM） C3758 CPU 2.20GHz
RAM	32GByte
NIC	Intel Coporation Ethernet Connection X553 1GbE （rev 11）

5. ネットワークトラフィック制御機能の予備実験

4.2 節で述べたシャドウバッファとバッファ間でのコピー処理時に行うトラフィック制御が可能か確認するために，ネットワークトラフィック制御機能の予備実験を行った．予備実験では，1Gbps Ethernet で接続されたクライアントと受信するサーバの 2 台のマシン間でパケットを送信を行い，トラフィック制御を行った場合と行っていない場合の送信レートを測定した．送信レートは，本提案手法のネットワークトラフィック制御機能で 1 秒あたり 20Mbytes に制限した．2 台のマシン間通信には，ネットワーク I/O の性能を測定するツールである iperf3[30] を使用し，ネットワークトラフィック制御機能で通信の上限値を設定していない状態と設定した状態の送信レートを比較した．クライアントの仕様を表 1，サーバの仕様を表 2，iperf3 の設定を表 3 に示す．

図 5 は，予備実験でのパケット送信レートの推移を示す．

表 3　iperf3 の設定

IP バージョン	IPv4
プロトコル	UDP
帯域幅	1000Mbits/sec
通信時間	30 秒

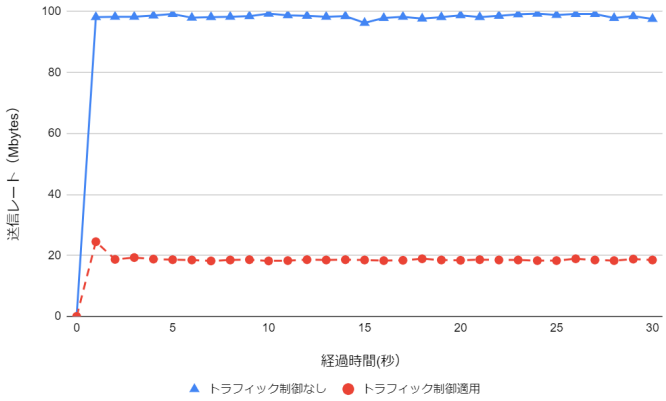

図 5　パケット送信レートの変化

実線がトラフィック制御を適用していない場合，破線がトラフィック制御を適用した場合である．縦軸は，1 秒あたりの送信レート（Mbytes），横軸は時間（秒）を表す．iperf3 で送信したパケットの送信レートを，1 秒あたり 20Mbytes に制限できていることが確認できる．そのため，本提案手法のシャドウバッファとバッファ間でのコピー処理時に行うトラフィック制御が可能であることが確認できた．

6. おわりに

本稿では，コンテナ型仮想化技術を用いたマルチテナント方式のクラウドサービスにおいて，クラウドサービス事業者が考慮する必要のあるネットワーク性能，トラフィックの公平性担保，インスタンスの利便性の要件についてまとめた．また，既存ネットワーク分離手法について，これらの 3 要件とコンテナ型仮想化技術の特徴である高集積なインスタンス配置の観点で整理した．軽量ハイパーバイザによって仮想ネットワークインタフェースを提供し，仮想ネットワークインタフェースに対してハイパーバイザからトラフィック制御する手法を提案した．

今後の課題として，本提案手法によるインスタンスの集積度への影響がある．本提案では，軽量ハイパーバイザによってネットワークインタフェースの提供とトラフィック制御のみを行う．そのため，仮想化なしの環境でインスタンスを提供する環境と比べ，集積度が低下する可能性があるため，集積度の差を評価していく必要がある．また，本稿の実装手法は，別マシンとの通信が対象であり，単一マシン内通信を行った場合はハイパーバイザでの処理がボトルネックになることが予測されるため，単一マシン内通信はカーネル内パケットフィルタリングとの併用による処理などを検討したい．

参考文献

[1] 松本亮介，栗林健太郎，岡部寿男：Web サーバの高集積マルチテナントアーキテクチャと運用技術，電子情報通信学会論文誌，Vol. 101-B, No. 1, pp. 16–30 (2018).
[2] Docker Inc.: Docker, `https://www.docker.com/` ((Accessed on 08/18/2020)).
[3] Matsumoto, R., Kondo, U. and Kuribayashi, K.: FastContainer: A Homeostatic System Architecture High-Speed Adapting Execution Environment Changes, *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1, pp. 270–275 (2019).
[4] Pahl, C.: Containerization and the PaaS Cloud, *IEEE Cloud Computing*, Vol. 2, No. 3, pp. 24–31 (2015).
[5] Bernstein, D.: Containers and Cloud: From LXC to Docker to Kubernetes, *IEEE Cloud Computing*, Vol. 1, No. 3, pp. 81–84 (2014).
[6] Zhang, W., Sharma, A., Joshi, K. and Wood, T.: Hardware-Assisted Isolation in a Multi-Tenant Function-Based Dataplane, *Proceedings of the Symposium on SDN Research*, SOSR '18, Association for Computing Machinery (2018).
[7] Shue, D., Freedman, M. J. and Shaikh, A.: Performance Isolation and Fairness for Multi-Tenant Cloud Storage, *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, USENIX Association, pp. 349–362 (2012).
[8] Jing Zhu, Dan Li, Jianping Wu, Hongnan Liu, Ying Zhang and Jingcheng Zhang: Towards bandwidth guarantee in multi-tenancy cloud computing networks, *2012 20th IEEE International Conference on Network Protocols (ICNP)*, pp. 1–10 (2012).
[9] IDC: Worldwide Server Operating Environments Market Shares, 2018: Overall Market Growth Accelerates, `https://www.idc.com/getdoc.jsp?containerId=US45656619` ((Accessed on 09/15/2020)).
[10] The Linux Foundation: 2017 State of Linux Kernel Development, `https://www.linuxfoundation.org/2017-linux-kernel-report-landing-page/` ((Accessed on 09/03/2020)).
[11] Zhao, Y., Xia, N., Tian, C., Li, B., Tang, Y., Wang, Y., Zhang, G., Li, R. and Liu, A. X.: Performance of Container Networking Technologies, *Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems*, HotConNet '17, Association for Computing Machinery, p. 16 (2017).
[12] Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K. and Casado, M.: The Design and Implementation of Open vSwitch, *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, USENIX Association, pp. 117–130 (2015).
[13] Dusia, A., Yang, Y. and Taufer, M.: Network Quality of Service in Docker Containers, *2015 IEEE International Conference on Cluster Computing*, pp. 527–528 (2015).
[14] Anderson, J., Hu, H., Agarwal, U., Lowery, C., Li, H. and Apon, A.: Performance considerations of network functions virtualization using containers, *2016 International Conference on Computing, Networking and Communications (ICNC)*, pp. 1–7 (2016).
[15] Cilium: cilium/cilium: eBPF-based Networking, Security, and Observability, `https://github.com/cilium/cilium` ((Accessed on 08/24/2020)).
[16] Cilium: eBPF - Introduction, Tutorials & Com-

munity Resources, https://ebpf.io/ ((Accessed on 10/22/2020)).

[17] Thomas, G.: KubeCon + CloudNativeCon Europe 2018, Cilium Accelerating Envoy with Linux Kernel, https://sched.co/EJAt ((Accessed on 08/26/2020)).

[18] Linux Foundation: Open vSwitch, https://www.openvswitch.org/ ((Accessed on 08/26/2020)).

[19] William, T.: Open vSwitch 2018 Fall Conference, Fast Userspace OVS with AF_XDP, https://www.openvswitch.org/support/ovscon2018/ ((Accessed on 08/26/2020)).

[20] Eelco, C. and William, T.: Open vSwitch and OVN 2019 Fall Conference, OVS with AF_XDP, what to expect, https://www.openvswitch.org/support/ovscon2019/ ((Accessed on 08/26/2020)).

[21] DPDK Project.: DPDK, https://www.dpdk.org/ ((Accessed on 08/18/2020)).

[22] Yang, M. and Huang, Y.: OVS-DPDK with TSO feature running under docker, *2018 International Conference on Information Networking (ICOIN)*, pp. 270–273 (2018).

[23] Yu, T., Noghabi, S. A., Raindel, S., Liu, H., Padhye, J. and Sekar, V.: FreeFlow: High Performance Container Networking, *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, HotNets '16, Association for Computing Machinery, p. 4349 (2016).

[24] Zhang, J., Lu, X. and Panda, D. K.: Performance Characterization of Hypervisor-and Container-Based Virtualization for HPC on SR-IOV Enabled InfiniBand Clusters, *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 1777–1784 (2016).

[25] 高野了成，工藤知宏，児玉祐悦：仮想計算機モニタ・バイパス型ネットワークに対する通信制御方式，研究報告システムソフトウェアとオペレーティング・システム（OS），No. 13, pp. 1–6 (2011).

[26] Intel Corporation: PCI-SIG SR-IOV Primer:An Introduction to SR-IOV Technolog, http://www.intel.com/content/www/us/en/pci-express/pci-sig-sr-iov-primer-sr-iov-technology-paper.html ((Accessed on 10/22/2020)).

[27] Ren, Y., Liu, G., Nitu, V., Shao, W., Kennedy, R., Parmer, G., Wood, T. and Tchana, A.: Fine-Grained Isolation for Scalable, Dynamic, Multi-tenant Edge Clouds, *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, USENIX Association, pp. 927–942 (2020).

[28] Nakamura, R., Sekiya, Y. and Tazaki, H.: Grafting Sockets for Fast Container Networking, *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems*, Association for Computing Machinery, pp. 15–27 (2018).

[29] Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E. and et al.: BitVisor: A Thin Hypervisor for Enforcing i/o Device Security, *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, Association for Computing Machinery, pp. 121–130 (2009).

[30] Dugan, J., Elliott, S., Mah, B. A., Poskanzer, J. and Prabhu, K.: iPerf - The TCP, UDP and SCTP network bandwidth measurement tool, https://iperf.fr/ ((Accessed on 10/26/2020)).