Vol.2021-OS-153 No.3
2021/7/20

KubernetesでのOVSフルオフロードによる低遅延化

加藤　純[1]　園田 雅崇[1]　白木 長武[1]　五木田 駿[1]　濱湊 真[1]

概要：Kubernetes はアプリケーションのクラウドネイティブ化のためにマイクロサービスアーキテクチャーを推奨しているがこのアーキテクチャーは通信回数が増えやすくレイテンシーの悪化や通信ソフトウェアの負荷増加を引き起こす．本稿では通信パケットの処理だけでなくその管理処理もハードウェアにオフロードできる NVIDIA BlueField を使って通信ソフトウェア OVS(Open vSwitch) のフルオフロードを提案，OVS のホストでの処理負荷をゼロにしつつハードウェアでパケット処理を行うことで低レイテンシーを実現する．またオフロードにより逆にレイテンシーが悪化するノード内通信向けにダイレクトフォワードも提案しオフロード自体に潜在する課題も解く．フルオフロードの設計は OVS ベースの CNI(Container Networking Interface) プラグイン Antrea をもとに行いノード内通信ではダイレクトフォワードで最大 55%，ノード間通信ではフルオフロードにより最大 57%のレイテンシーを削減した．

キーワード：Kubernetes, OVS, Full Offload, BlueField, Antrea

OVS Full Offload on Kubernetes for Low Latency

Jun Kato[1]　Masataka Sonoda[1]　Osamu Shiraki[1]　Shun Gokita[1]　Makoto Hamaminato[1]

Abstract: Kubernetes recommends microservice architecture to make applications cloud-native, but this architecture can easily increase the number of communications, which lead to lower latency and increased load on the communication software. This paper proposes full offloading of OVS (Open vSwitch) communication software using NVIDIA BlueField, which can offload not only communication packet processing but also its management processing to the hardware, and realize low latency by hardware offload while reducing processing load of OVS at the host to zero. We also propose direct forward for intra-node communication where the latency is degraded by offloading itself. The full offload was designed from Antrea, an OVS-based CNI (Container Networking Interface) plug-in, and reduces latency by up to 55% with direct foward for intra-node communications and up to 57% with full offload for inter-node communications.

Keywords: Kubernetes, OVS, Full Offload, BlueField, Antrea

1. はじめに

パブリッククラウドやプライベートクラウドでのコンテナオーケストレーションシステムとして Kubernetes [1] はデファクトスタンダードな地位を確立している [2]．Kubernetes ではクラウドネイティブなアプリケーションを低コストで開発・運用するためにマイクロサービスアーキテクチャー [3–6] を推奨しており，このアーキテクチャーに基づいたアプリケーションでは従来のモノリシックアーキテクチャーと比較して機能単位での俊敏な開発や柔軟なスケールアウト，障害分離による耐障害性を容易に実現することができる．マイクロサービスアーキテクチャーはマイクロサービスと呼ばれる小さなサービスを基本単位として各マイクロサービスが API によって連携するアーキテクチャーでそのマイクロサービスを実現するために Kubernetes では 1 つ以上のコンテナを含む Pod を最小の基本単位として定義している．ユーザーはコンテナ間で密結合が必要な場合は Pod 内にまとめ Pod 間は疎結合を保つことだけに注意すれば Pod をマイクロサービスと見立てることができ，基盤である Kubernetes のフレームワーク

[1] 富士通株式会社
Fujitsu Limited

に便乗して簡単にアプリケーションをクラウドネイティブ化することができる．

マイクロサービスアーキテクチャーが浸透してマイクロサービスの細粒化によりその個数が増えてくるとそれらの間の通信回数も増加してしまうアーキテクチャー上の課題も浮き彫りになった [7,8]．Kubernetes では CNI(Container Networking Interface) [9] 準拠のプラグイン [10–14] をもとに通信を行うがこれら CNI プラグインは Netfilter [15] や eBPF(extended Berkeley Packet Filter) [16]，OVS(Open vSwitch) [17] 等の通信ソフトウェアで実装されている．そのため通信回数が増えるとアプリケーション全体のレイテンシーが増加するだけでなくこれら通信ソフトウェアの負荷も増えてしまいユーザーが自由に使えるリソース量が低下する．クラウド上のインタラクティブなオンラインサービス分野ではアプリケーション全体で数十ミリ秒程度，マイクロサービス単位だとミリ秒未満の低レイテンシーが求められており通信部分のレイテンシー増加は大きな課題である [18]．

この課題を解決するためいくつかの CNI プラグインではそれら通信ソフトウェアのハードウェアオフロードをサポートしている [10,12,13]．根本原因である通信回数を削減することはアーキテクチャ上困難なので，ハードウェアオフロードを活用してレイテンシーと通信ソフトウェアの負荷そのものを減らすことが狙いである．しかし既存の CNI プラグインでオフロードできるのはパケット処理を行うデータプレーンのみであり，その管理を行うコントロールプレーンも含めた通信ソフトウェア全てのオフロードはサポートしていない．またオフロードにより逆にレイテンシーが増加する場合もあるが既存の CNI プラグインではその場合を考慮せず常にオフロードを行っている．このように現状のハードウェアオフロードにはまだレイテンシーと通信ソフトウェアの負荷を減らす余地がある．

本稿ではマイクロサービスアーキテクチャーの課題を解決するために CNI プラグインが使う通信ソフトウェアを全てハードウェアにオフロードするフルオフロードを提案する．フルオフロードを実現できるコモディティなハードウェアと通信ソフトウェアしてそれぞれ NVIDIA BlueField [19] と OVS を選定，OVS ベースの CNI プラグインである Antrea [10] をもとにフルオフロードの設計を行った．また OVS のデータプレーンオフロードによりノード内通信レイテンシーが増加することを明らかにし，それを解決するダイレクトフォワードも提案する．性能評価ではダイレクトフォワードによりノード内通信のレイテンシーを最大 55%削減，またフルオフロードによりホストでの OVS の処理負荷をゼロにしつつノード間通信のレイテンシーを最大 57%削減，99 パーセンタイル値でも 1 ミリ秒以下の低レイテンシーを実現した．

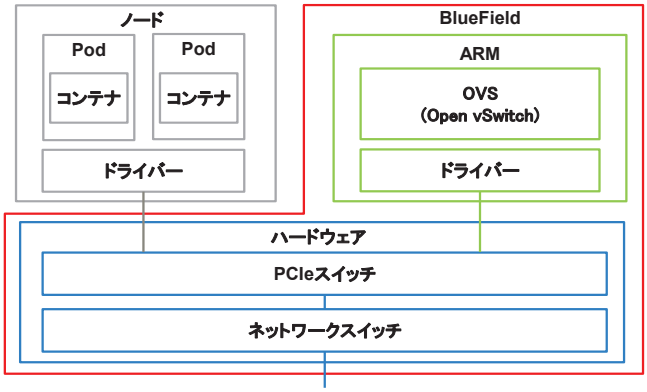

図 1 NVIDIA BlueField の構成

2. 関連研究

ハードウェアオフロードをサポートしている主流の通信ソフトウェアとして OVS [17] がある．OVS は SDN(Software Defined Networking) [20] を実現する OpenFlow [21] ベースの仮想スイッチでありパケット処理を行うデータプレーンとその制御を行うコントロールプレーンの 2 つから構成される．OVS のデータプレーンはカーネル空間と DPDK [22] を使ったユーザー空間の 2 種類の実装があり，前者は TC Flower，後者は rte_flow を使ってデータプレーンのオフロード [23–25] ができる．OVS ベースの CNI プラグイン Antrea [10] や OVN-Kubernetes(Open Virtual Network) [13] はデータプレーンにカーネル空間の実装を採用して TC Flower を使ったオフロードもサポートしているが，どちらもコントロールプレーンはノード上のユーザー空間で動作しており OVS の全てをオフロードできていない．OVS のコントロールプレーンはパケットのオフロード設定を行うだけではなく，その管理のために定期的にハードウェア上のオフロード設定と統計情報を全て吸い出しオフロード設定を再評価する高負荷な処理も行う [26]．この処理時間はオフロード数に比例するが，Pod 数が多くなるマイクロサービスアーキテクチャーではオフロードの数も増えやすくコントロールプレーンの負荷を無視できない．

図 1 の NVIDIA の BlueField [19] に代表される SmartNIC は OVS のオフロード機能のほかに CPU やメモリ等のコンピューティング機能を搭載した NIC である．このコンピューティング機能はホストで行っていた処理のオフロードに活用されており，E3 [6] はマイクロサービスを低消費電力な SmartNIC にオフロードすることで電力効率を改善，iPipe [27] は SmartNIC の CPU 使用率を指標に SmartNIC へのオフロード量を動的に決定するスケジューラーを提案しホストと SmartNIC 両方での利用効率を向上，FairNIC [28] は SmartNIC でテナントごとの仕様に応じたリソース管理を行うことでマルチテナントでの通信性能の分離を実現した．BlueField はこのコンピューティング機

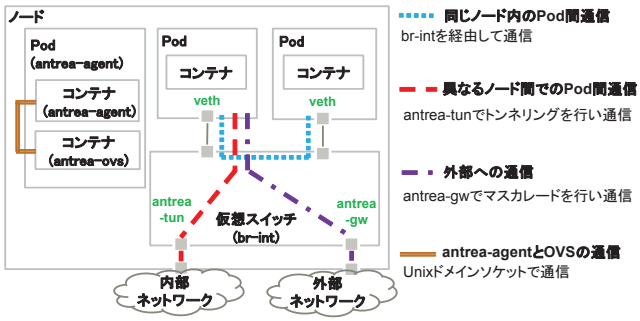

図 2 Antrea の構成 (従来，オフロードなし)

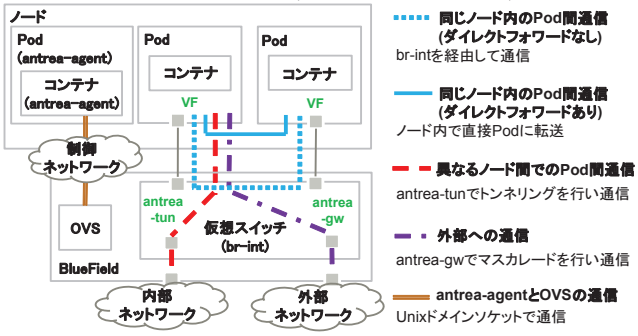

図 3 Antrea の構成 (フルオフロード)

能を使って OVS のコントロールプレーンを SmartNIC で動作させることができるコモディティ製品であるが既存の CNI プラグインはまだ BlueField をサポートしていない．

Netronome の SmartNIC である Agilio [29] は OVS の他に eBPF のハードウェアオフロードをサポートしており [30,31]，eBPF ベースの CNI プラグイン Cilium [12] はこの機能を活用してデータプレーンのオフロードを実現している．eBPF は C に近い言語でデータプレーンの処理を記述しコンパイラで eBPF のバイトコードに変換，システムコールを通して変換したバイトコードをカーネル内にロードし JIT コンパイラがネイティブコードに変換してから記述した処理を各種イベント時に呼び出す．Agilio では JIT コンパイラがホスト CPU ではなく Agilio のネイティブコードに変換することでデータプレーンのオフロードが可能であるが，コントロールプレーンはホスト上で動作するため eBPF の全てをオフロードできていない．

3. Antrea の OVS フルオフロード

本稿では OVS のデータプレーンオフロードに加えて SmartNIC のコンピューティング機能を活用してコントロールプレーンもオフロードするフルオフロードを行う．SmartNIC にはコモディティ製品でコントロールプレーンをオフロードできる NVIDIA の BlueField，通信ソフトウェアとしては BlueField がハードウェアオフロードをサポートしている OVS，CNI プラグインには OVS ベースでデータプレーンのオフロードに対応している Antrea を採用した．OVS ベースの CNI プラグインには OVN-Kubernetes もありその CNI プラグインでは BlueField のフルオフロードも仕様上はサポートしているが筆者らが予備評価したところエラーが発生したため実用段階ではないと判断，OVN の仮想ネットワークを介さず OVS を直接操作するためきめ細やかな開発が可能でトラブルシューティングも容易な Antrea を選択した．以下では図 2 のオフロードがない従来の Antrea の構成と図 3 のフルオフロード時の構成を対比させながらフルオフロードの特徴を述べる．

3.1 コントロールプレーン

Antrea は Kubernetes の DaemonSet を用いて antrea-agent の Pod を各ノードに 1 つずつ配置し，Kubernetes から Pod の作成または削除のイベントを受け取るとこの antrea-agent が受け取ったイベントの処理を行う．従来の構成では antrea-agent の Pod は同名の antrea-agent コンテナと antrea-ovs コンテナの 2 つから構成されこの 2 つのコンテナは共通にマウントしている Unix ドメインソケットを経由して接続するが，フルオフロードの構成では OVS がノード上のコンテナではなく BlueField 上で動作するためファイルシステムのマウントではなく Unix ドメインソケットのフォワーディングによって制御ネットワーク経由で接続する．Unix ドメインソケットは antrea-agent から OVS が管理する仮想スイッチ br-int を操作するためのインターフェースで Pod と br-int の接続や Pod が通信するためのデータプレーン設定はこのソケットを経由して行う．

Antrea は従来では Pod と br-int を仮想ポート (veth) 形式で接続するが，オフロード時は br-int を switchdev モードに設定し SR-IOV(Single Root I/O Virtualization) の VF(Virtual Function) を使って接続する [24,32]．BlueField に対してノード内の Pod 宛にパケットが届いた場合，オフロード設定されている場合は接続している Pod に直接，オフロード設定されていない場合は一度 Representor ポートを経由して BlueField 上の OVS にパケットを転送してそこでソフトウェアでパケット処理方法を決定後にオフロード設定してからハードウェアで Pod にパケットを転送する．br-int には Pod につながるポートの他に外部通信用の antrea-gw と他ノード上の Pod とのトンネリングに使う antrea-tun の 2 つの仮想ポートがあり，それぞれ外部またはノード間をつなぐ内部ネットワークにつながっている物理ポートと接続する．

3.2 データプレーン

Antrea の IPAM(IP Address Management) はノードごとに 1 つのサブネットを割り当て Pod にはそのサブネットから IP アドレスを割り当てる．そのため Pod から通信する場合は宛先のサブネットが (1) 自ノードの場合はノード内，(2) 他ノードの場合はノード間，(3)IPAM が払いだしたサブネットでない場合は外部，の 3 通りの通信がある．まずノード内通信に着目すると従来では名前の通りノード

内に閉じた転送処理を行うがデータプレーンのオフロードを使用するだけの場合は VF を経由して一度ノード外の BlueField にパケットを送りそこでハードウェアオフロードされたデータプレーンが送信元と同じノード内の Pod に対して転送処理を行う．ハードウェアオフロードで転送処理のレイテンシーを削減できるが，それよりも BlueField に対してパケットを送受信してしまうオーバーヘッドが大きくオフロードにより逆にレイテンシーが悪化する．そこで本稿では 4 章で述べるダイレクトフォワードを用いてノード内通信の場合はハードウェアオフロードを使わずに直接 Pod に転送することでこの性能低下を回避する．

データプレーンの設定を行う antrea-agent は従来であれば起動時に自ノードの物理ポートのみを検出していたが，フルオフロードではノード間通信と外部通信のために BlueField の物理ポートを代わりに検出する．Pod に割り当てられた IP アドレスはノード内でのみ有効なサブネットから払いだしたものでありこれだけではノード外と通信ができない．そのためノード間通信ではトンネリングヘッダーを追加，外部通信はマスカレードによる送信元書き換えを行いノード外とは物理ポートの IP アドレスをもとにした通信を行う．このために antrea-agent は起動時にノードの物理ポートを検出しそれをもとにデータプレーンの設定を行っていたがフルオフロードではノードではなく BlueField の物理ポートの IP アドレスをもとにトンネリングやマスカレードを行う．このように Pod が動作するノードではなく BlueField のネットワークをもとにノード外と通信を行うため物理ポート以外にも外部通信のためのルーティング設定などノード外と関わるネットワーク設定全てを BlueField のネットワークをもとにして行う．

4. ノード内通信のダイレクトフォワード

OVS のデータプレーンオフロードではすべてのパケットを一度 BlueField に転送させそこでハードウェア処理を行うが BlueField に転送するため少なくとも一度はホストと BlueField の間でのパケットの送受信が必要である．ノード外通信であれば従来も物理ポートでパケットの送受信を行うためペナルティはないが，ノード内通信に関しては従来はノード内に閉じてパケットを転送するだけなのでこのパケットの送受信処理は追加のオーバーヘッドでありこれが原因でノード内通信はオフロードした方が逆にレイテンシーが悪化することを 3 章で述べた．本章ではこれを解決するためにノード内通信となるパケットであれば BlueField に転送させず従来と同様にそのまま宛先の Pod に転送するダイレクトフォワードについて述べる．

ダイレクトフォワードはノード内通信を識別するためにパケットの宛先 MAC アドレスを利用する．Kubernetes が Pod を作成または削除する際に antrea-agent はイベントを受け取りそれをもとにネットワーク構成の更新を行うがその中には通信に必要な Pod の MAC アドレスが含まれる．そこで antrea-agent がこのイベントを受け取るとダイレクトフォワード対象となる MAC アドレスとして登録または削除を行いパケットの宛先 MAC アドレスと登録されている MAC 一覧を比較することでノード内通信を識別する．このようにノード内通信を識別するためにはパケットとして宛先 MAC アドレスが確定していることが前提であるが，ダイレクトフォワードでは最終的なパケットが組み立てられ送信キューからデバイスドライバーに対してパケットが送られる際にこの識別処理を行うことでその前提を満たし正しくノード内通信を識別できる．

宛先 MAC アドレスを比較してノード内通信を検出すると送信用のパケットをデバイスドライバーが受信処理をした直後と同じパケットになるように書き換えデバイスドライバー以降の受信処理を行う．パケットの書き換えや後続の受信処理を呼び出すために受信を行うはずであったデバイスの情報が必要であるが，antrea-agent からの MAC アドレス登録時に MAC アドレスと紐づけてデバイスの情報も登録しておくことでダイレクトフォワードで転送されてきたパケットの送信先 MAC アドレスをキーにして取得することができる．このようにパケットをデバイスドライバーに渡すことなく直接受信処理を呼び出すことで従来と同じく BlueField へのパケットの送受信なくノード内通信を行うことができる．

表 1 ハードウェアの諸元

サーバー	Supermicro SYS-2028U-TN24R4T+ 3 台
CPU	Intel Xeon E5-2697 v4 2.30GHz 36 コア (HTT 有効，2 ソケット)
メモリー	PC4-17000 2133MHz DDR4 64GiB 8 枚
NIC	Intel Ethernet Controller 10G X550T 4 ポート
スイッチ	SH-E514TR1 1 台
ポート	100/1000/10GBase-T 12 ポート 10GBase-SR/LR/CR 2 ポート
BlueField	MBF2H516A-CEEOT 1 枚
CPU	ARMv8 Coretex-A72 8 コア
メモリー	DDR4 SDRAM 16GiB 1 枚
ポート	100Gb Ethernet QSFP56 2 ポート
コネクター	PCIe Gen 4.0(16GT/s)x16 レーン

5. 性能評価

表 1 に示すサーバー 3 台で Kubernetes クラスターを構成しうち 1 台に BlueField を搭載, 残りの 2 台はサーバー内蔵の NIC を 10G Ethernet スイッチにつないでクラスター内通信用の 10G ネットワークを構築し図 4 のようにノード内とノード間通信の性能評価を行った．本評価ではノード単体をフルオフロードした際の効果を検証するため 1 台だけに BlueField を搭載したが今後全ノードで BlueField を搭載した場合の効果も検証する予定である．Kubernetes

表 2　ソフトウェアの諸元

サーバー	
Linux	4.18.0-240.10.1.el8_3.x86_64 (CentOS 8)
OVS	2.13.1
Kubernetes	v1.12.0
CRI	CRI-O v1.20.0
CNI	Antrea v0.12.0
BlueField	
Linux	5.4.44-mlnx.9.gdffcb36f52b7
OVS	2.13.1
評価ツール	
レイテンシー測定	netperf 2.7.0
CPU 負荷	stress-ng 0.12.01
ネットワーク負荷	iperf3 3.5

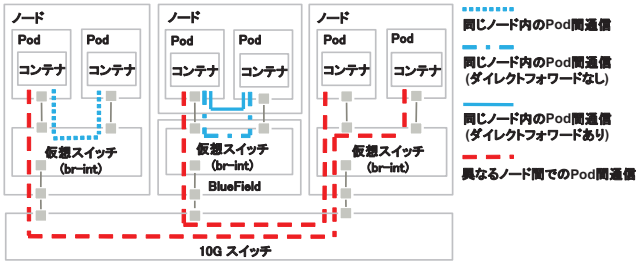

図 4　評価構成

表 3　ノード内通信の 128B UDP レイテンシー (背景負荷なし)

フルオフロード	なし	あり	あり
ダイレクトフォワード		なし	あり
最小値 [usec]	12	18	9
平均値 [usec]	15.64	25.25	13.04
50 パーセンタイル値 [usec]	15	24	13
90 パーセンタイル値 [usec]	16	29	13
99 パーセンタイル値 [usec]	31	45	30

クラスターの構築および性能評価には表 2 のソフトウェアを使い，BlueField を搭載したノードでは 4 章で述べたダイレクトフォワードを実装したカーネルを用いた．CNI プラグインには 3 章で述べたフルオフロードを実装した Antrea を用い，ノード間通信のトンネリングプロトコルには BlueField がハードウェアオフロードをサポートしている VXLAN [33] を採用した．

5.1　背景負荷なしの通信レイテンシー

表 3 は従来とフルオフロードの 2 種類の構成で同一ノード内に Pod を 2 つ立ち上げ他に負荷をかけない状態で Pod 間での 128B UDP の RTT(Round Trip Time) を計測した結果である．この結果から 3 章で述べたようにフルオフロードでもダイレクトフォワードがない場合は従来よりも 1.5 倍近くレイテンシーが悪化しておりデータプレーンのオフロードだけでは逆に性能が低下することが分かる．またこの時のパケット送受信で発生する割り込み回数を調べると従来は秒間あたり 5.4K 回だったのに対してオフロードを行うと 120K 回と 20 倍近く増加しておりホストの負荷削減を目指すオフロードが逆に処理負荷を増やす結果となった．これに対してダイレクトフォワードを有効にするとレイテンシーを最大 55%削減，割り込み回数についても 95%削減して 5.4K 回と従来と同程度のレイテンシーと割り込み回数を達成した．これによりダイレクトフォワードでデータプレーンのオフロードによるレイテンシー低下と処理負荷の増加を回避することができペナルティなくハードウェアオフロードのメリットを享受することができる．

表 4　ノード間通信の 128B UDP レイテンシー (背景負荷なし)

フルオフロード	なし	あり
最小値 [usec]	49	36
平均値 [usec]	60.00	47.96
50 パーセンタイル値 [usec]	53	43
90 パーセンタイル値 [usec]	77	62
99 パーセンタイル値 [usec]	104	85

表 4 は同一ノードではなく異なるノードに Pod を 1 つずつ立ち上げ他に負荷をかけない状態で Pod 間の 128B UDP の RTT を計測した結果である．ノード間通信は VXLAN で行われるため物理ポートから送信する場合にはカプセル化を行う encap，受信する場合にはカプセル化されたパケットから元のパケットを取り出す decap 処理を行う．従来はこれらの処理はソフトウェアで行っていたがフルオフロードでは encap と decap の両方ともハードウェアで行うため最小値から 99 パーセンタイル値までの全てでレイテンシーが低下し最大で 20%近い削減となった．このようにハードウェアオフロードはノード間通信での低レイテンシー化とホストでの処理負荷削減に直接寄与している．

5.2　背景負荷ありの通信レイテンシー

本節では実環境を想定して他の Pod を動作させ背景負荷がかかっている状態での評価を行う．背景負荷としては通信処理に関係する CPU 負荷とネットワーク負荷の 2 つを想定しこれらの負荷量を変えた場合の通信レイテンシーの推移を調べることでフルオフロードが有効である範囲を明らかにする．CPU 負荷は負荷が偏らないように指定分の負荷を各ノードの CPU コアごとにかけネットワーク負荷は各ノードに対して数珠つなぎに指定された帯域分の TCP 負荷をかけ各ノードでの送信と受信の帯域消費量が同じになる状態で評価を行った．

図 5 は背景負荷をかけた状態で従来とダイレクトフォワードを有効にしたフルオフロード構成での同一 Node 内の Pod での 128B UDP レイテンシーである．背景負荷がない場合のノード内通信はどちらの構成も同程度のレイテンシーであったが，負荷をかけてもレイテンシーは大きく変化せず有意な差がなかった．これはノード内通信の場合には Pod から送信処理が呼ばれた延長で受信処理まで CPU を手放すことなく一直線で行うため他の Pod が CPU

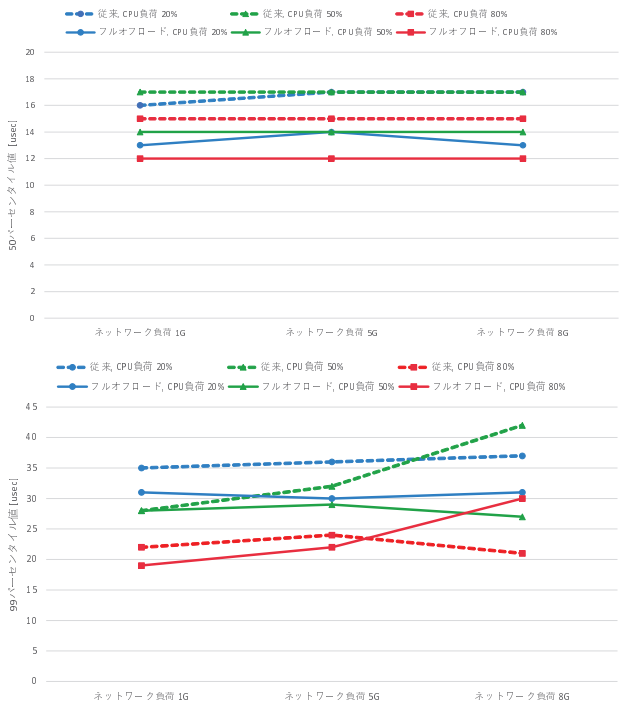

図 5 ノード内通信の 128B UDP レイテンシー
(背景負荷あり, 50 または 99 パーセンタイル値)

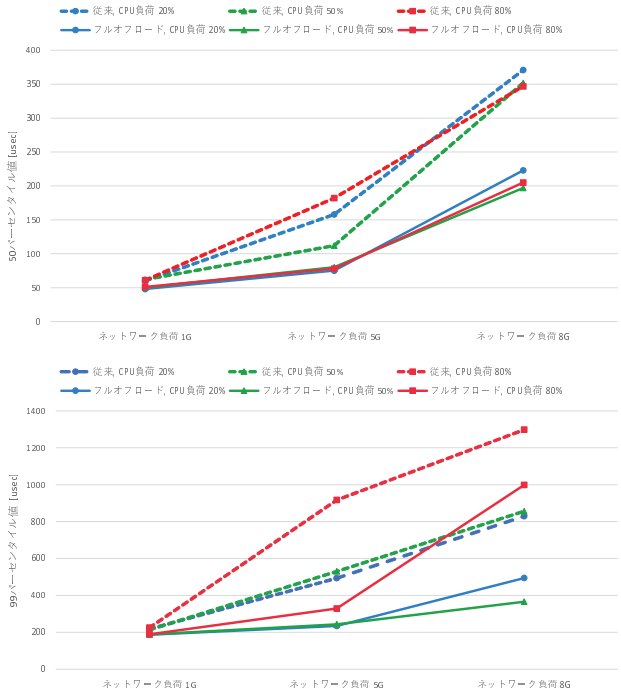

図 6 ノード間通信の 128B UDP レイテンシー
(背景負荷あり, 50 または 99 パーセンタイル値)

を使っていてもその影響をあまり受けないこと，またノード内に閉じた処理のためネットワーク負荷の影響を受けないことが要因だと考えられる．このようにダイレクトフォワードが有効なノード内通信では背景負荷によらず数十マイクロ秒の安定したレイテンシーであることが分かる．

異なるノードに Pod を 1 つずつ立ち上げ背景負荷をかけた状態での 128B UDP レイテンシーを示したのが図 6 である．ノード間通信の場合は背景負荷，特にネットワーク負荷の増加に伴うレイテンシー増加が著しく従来の場合はネットワーク負荷 8G の場合には 50 パーセンタイル値で 347 マイクロ秒，99 パーセンタイル値で 1298 マイクロ秒まで増加する．CPU 負荷についても 50 パーセンタイル値では大きな影響はないものの 99 パーセンタイル値に着目すると CPU 負荷 80%の場合にレイテンシーが大きく上振れネットワーク負荷 5G でも 917 マイクロ秒と 1 ミリ秒に近くに跳ね上がっている．これに対してフルオフロードでは背景負荷によるレイテンシー増加はあるもののハードウェアオフロードによりその増加割合は従来よりも小さくなっており 50 パーセンタイル値で最大 57%レイテンシーを削減し 99 パーセンタイル値でも 1 ミリ秒以下のレイテンシーとなった．このようにフルオフロードは背景負荷があるノード間通信で最も恩恵が大きい．

マイクロサービスアーキテクチャーで Pod が細分化されその数が増えてくると Pod 間の通信をすべてノード内に完結して行える可能性は小さくなり，逆に一回以上ノード間通信が発生する可能性が増える．その場合のボトルネックは背景負荷に影響されずに数十マイクロ秒で完了するノード内通信ではなく背景負荷に影響されやすく 1 ミリ秒近くにまでレイテンシーが悪化するノード間通信である．フルオフロードはこのボトルネックであるノード間通信時に最もその効果を発揮しレイテンシーと通信ソフトウェアの負荷の両方を大きく削減できていることからマイクロサービスアーキテクチャーが抱える課題を解決する有効な手法であると考えられる．

6. おわりに

マイクロサービスアーキテクチャーでは通信回数が増えやすいためアプリケーション全体のレイテンシーが低下しやすく通信ソフトウェアの負荷も増加しやすくなる課題があった．本稿では通信ソフトウェアの 1 つである OVS のフルオフロードを提案，データプレーンをハードウェアにオフロードすることでレイテンシーを下げるだけでなく BlueField を使ってコントロールプレーンもオフロードすることで OVS のホストからの解放を実現した．また従来のデータプレーンのオフロードではノード内通信が逆に悪化することを示しそれを解決するダイレクトフォワードも提案した．評価ではダイレクトフォワードによりノード内通信はレイテンシーを最大 55%削減するだけでなく背景負荷に影響されることなく安定した低レイテンシーとなること，ノード間通信ではレイテンシーを最大 57%削減し負荷をかけた状態でも 99 パーセンタイル値で 1 ミリ秒以下の低レイテンシーとなることを示した．

今後の課題として OVN-Kubernetes でのフルオフロードの評価と全ノードでフルオフロードを行った場合の評価

がある．OVN-Kubernetes については OSS(Open Source Software) でフルオフロードの開発が続けられており筆者らが遭遇したエラーが解決している可能性もある．コミュニティの動向を注視し実用レベルにまで安定化した段階でその評価を行う予定である．また本評価ではノード 1 台だけをフルオフロードしたがそのノードの通信先である対向ノードでは通信処理はハードウェアオフロードされておらずソフトウェア処理となっている．そのため対向ノードでもフルオフロードすることができればレイテンシーをさらに削減することができると考えられ今後はその評価も行う予定である．

謝辞 設計と評価にあたり富士通 ヘテロスケーラープロジェクトのみなさまから多数のアドバイス・コメント・フィードバックをいただき大変お世話になりました．ここに深く感謝の意を表します．また富士通の兵頭 和樹氏にはダイレクトフォワードのアイデアレビューから設計・実装の相談まで本研究を進めるにあたって多大なご協力およびご支援をいただきました．心から感謝を申し上げます．

参考文献

[1] The Kubernetes Authors: Kubernetes, `http://kubernetes.io`.
[2] Bernstein, D.: Containers and Cloud: From LXC to Docker to Kubernetes, *IEEE Cloud Computing*, Vol. 1, No. 3, pp. 81–84 (2014).
[3] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R. and Safina, L.: Microservices: Yesterday, Today, and Tomorrow, *Present and Ulterior Software Engineering*, pp. 195–216 (2017).
[4] Abdollahi Vayghan, L., Saied, M. A., Toeroe, M. and Khendek, F.: Deploying Microservice Based Applications with Kubernetes: Experiments and Lessons Learned, *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pp. 970–973 (2018).
[5] Abdollahi Vayghan, L., Saied, M. A., Toeroe, M. and Khendek, F.: Microservice Based Architecture: Towards High-Availability for Stateful Applications with Kubernetes, *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, pp. 176–185 (2019).
[6] Liu, M., Peter, S., Krishnamurthy, A. and Phothilimthana, P. M.: E3: Energy-Efficient Microservices on SmartNIC-Accelerated Servers, *2019 USENIX Annual Technical Conference* (2019).
[7] Qi, S., Kulkarni, S. G. and Ramakrishnan, K. K.: Understanding Container Network Interface Plugins: Design Considerations and Performance, *2020 IEEE International Symposium on Local and Metropolitan Area Networks*, pp. 1–6 (2020).
[8] Qi, S., Kulkarni, S. G. and Ramakrishnan, K. K.: Assessing Container Network Interface Plugins: Functionality, Performance, and Scalability, *IEEE Transactions on Network and Service Management*, Vol. 18, No. 1, pp. 656–671 (2021).
[9] Cloud Native Computing Foundation: CNI – the Container Network Interface, `https://github.com/containernetworking/cni`.
[10] Antrea Authors: Antrea, `https://antrea.io`.
[11] Tigera: Calico, `https://www.projectcalico.org`.
[12] Cilium Authors: Cilium, `https://cilium.io`.
[13] OVN (OVN Virtual Network) Authors: OVN-Kubernetes, `https://github.com/ovn-org/ovn-kubernetes`.
[14] Weaveworks: Weave, `https://www.weave.works`.
[15] Netfilter Authors: Netfilter, `https://netfilter.org`.
[16] eBPF Authors: eBPF (extended Berkely Packet Filter), `https://ebpf.io/`.
[17] 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* (2015).
[18] Jia, Z. and Witchel, E.: Nightcore: efficient and scalable serverless computing for latency-sensitive, interactive microservices, *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 152–166 (2021).
[19] NVIDIA: BLUEFIELD DPU (Data Processing Unit), `https://www.nvidia.com/ja-jp/networking/products/data-processing-unit/`.
[20] Kirkpatrick, K.: Software-Defined Networking, *Communications of the ACM*, Vol. 56, No. 9, pp. 16–19 (2013).
[21] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. and Turner, J.: OpenFlow: Enabling Innovation in Campus Networks, *ACM SIGCOMM*, Vol. 38, No. 2, pp. 69–74 (2008).
[22] The Linux Foundation: DPDK (Data Plane Development Kit), `https://www.dpdk.org/`.
[23] Horman, S.: TC Flower Offload, *NetDev* (2017).
[24] Efraim, R. and Gerlitz, O.: Using SR-IOV Offloads with Open-vSwitch and similar applications, *Netdev*, Vol. 1 (2016).
[25] 兵頭和樹, 清水貴志, 深野晴久, 麻野克仁, 松浦康道：FPGA 内蔵 CPU を用いた OVS オフロードの開発と評価，信学技報，Vol. 119, No. 196, pp. 33–38 (2019).
[26] Aliyu, A. L., Bull, P. and Abdallah, A.: Performance Implication and Analysis of the OpenFlow SDN protocol, *2017 31st International Conference on Advanced Information Networking and Applications Workshops*, IEEE, pp. 391–396 (2017).
[27] Liu, M., Cui, T., Schuh, H., Krishnamurthy, A., Peter, S. and Gupta, K.: Offloading Distributed Applications onto SmartNICs Using IPipe, *ACM SIGCOMM* (2019).
[28] Grant, S., Yelam, A., Bland, M. and Snoeren, A. C.: SmartNIC Performance Isolation with FairNIC: Programmable Networking for the Cloud, *ACM SIGCOMM* (2020).
[29] Netronome: Agilio, `https://www.netronome.com/products/smartnic/`.
[30] Kicinski, J. and Viljoen, N.: eBPF Hardware Offload to SmartNICs: cls bpf and XDP, *NetDev*, Vol. 1 (2016).
[31] Vieira, M. A. M., Castanho, M. S., Pacífico, R. D. G., Santos, E. R. S., Júnior, E. P. M. C. and Vieira, L. F. M.: Fast Packet Processing with EBPF and XDP: Concepts, Code, Challenges, and Applications, *ACM Computing Surveys*, Vol. 53, No. 1 (2020).
[32] Lesokhin, I., Eran, H. and Gerlitz, O.: Flow-based tunneling for SR-IOV using switchdev API, *NetDev* (2016).
[33] RFC Edtior: Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks, RFC 7348.