様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

令和　２ 年　 ５ 月　 ８ 日現在

機関番号：１２６０１
研究種目：若手研究
研究期間：2018～2019
課題番号：１８Ｋ１８０３４

研究課題名（和文）コンテナ仮想化技術におけるネットワーク高速化手法の研究

研究課題名（英文）Improving Packet Transport in Container Virtualization

研究代表者
中村　遼（Nakamura, Ryo）

東京大学・情報基盤センター・助教

研究者番号：９０８０４７８２

交付決定額（研究期間全体）：（直接経費）　　2,300,000円

研究成果の概要（和文）：本研究では、コンテナ仮想環境におけるネットワーク性能の低下を回避する技術を開発した。コンテナ仮想環境では、コンテナ内部のアプリケーションが外部と通信する際に、データはコンテナとホストOSで2度ネットワークスタックを通過する。これによる性能低下を回避するため、Socket Graftingと呼ばれる、コンテナ内部のソケット層とホストOSのソケット層を接続する新しいデータ通信チャネルを設計、実装した。Socket Graftingによってコンテナ環境におけるネットワーク性能を物理ホストと同程度まで改善した。

研究成果の学術的意義や社会的意義
本研究では、コンテナ仮想環境におけるネットワーク性能の向上を実現した。また提案手法は高速パケットI/O技術による高速化のようにネットワークスタックそのものを再実装する必要がなく、既存の汎用OSの成熟したネットワークスタック実装をそのまま利用することが可能である。マイクロサービスアーキテクチャやサーバレスコンピューティングなどコンテナ仮想化を前提とした新しい計算基盤の普及が進む中で、本研究はそうした基盤のネットワークの高速化を実現可能な、実践的なコンテナネットワーク高速化技術である。

研究成果の概要（英文）：In this study, we have proposed a new data communication channel, called socket-grafting, for avoiding the network performance degradation in container-based virtualized environments. In container virtualization, data sent from an application in a container needs to be processed by two network stacks in the container and the host OS. This long data path causes degradation of network performance of containerized applications. To avoid the degradation, we designed and implemented socket-grafting that grafts sockets in containers onto sockets in host network stacks for bypassing container network stacks. The experiments with containerized applications show that socket-grafting achieves throughput and latency comparable with the native hosts.

研究分野：ネットワーク仮想化

キーワード：コンテナネットワーク　仮想化技術　ソケット　Linux

※科研費による研究は、研究者の自覚と責任において実施するものです。そのため、研究の実施や研究成果の公表等については、国の要請等に基づくものではなく、その研究成果に関する見解や責任は、研究者個人に帰属されます。

様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９（共通）

１．研究開始当初の背景

増え続けるサービストラフィックを処理するために、Web サービスなど多種多様な情報システムは、より高い性能を実現すると同時に、複雑化するシステムを効率的にメンテナンスし運用できる柔軟性を求められている。このような背景から、近年コンテナ技術が注目されている。コンテナ技術とは、物理ホスト上で、ネットワークやファイルシステムといったホスト OS の資源を名前空間を用いて分離し、ホスト OS 内に独立したアプリケーション実行環境を構築する技術である。この独立した実行環境はコンテナと呼ばれ、ユーザはコンテナ内で自由にライブラリを追加し、アプリケーションを実行することができる。

類似の仮想化技術である仮想マシンに対して、コンテナを用いる利点は以下の 2 点が挙げられる。

- オーバーヘッドの小ささ: コンテナは、仮想マシンのようなハイパーバイザやゲスト OS の処理が必要無いため、より小さなオーバーヘッドで動作する。そのため、同じ物理ホスト上でより多くの仮想環境を実行することができる。
- 実行環境のパッケージングによる運用の簡素化: アプリケーションとその実行に必要なライブラリをコンテナイメージとしてパッケージングすることができる。アプリケーションをデプロイする際はコンテナイメーを実行するだけとなり、環境構築の手間を減らすことができる。

こうした理由により、近年コンテナ技術を用いた情報システムの構築が普及しつつある。

Linux Container や Docker といったコンテナ管理ソフトウェアは、ホスト OS 内に構築されるコンテナに外部ネットワークへの接続性を提供するため、NAT を利用した仮想ネットワークを構成する。図 1 に、NAT を用いた一般的なコンテナネットワークの構成を示す。各コンテナは名前空間で分離された TCP/IP スタックを持ち、独立した IP アドレスや経路表を保持する。コンテナとホスト OS の TCP/IP スタックは仮想的な Network Interface Card (vNIC)を介して接続され、ホスト OS の TCP/IP スタックが各コンテナに対して NAT ルータとして振る舞う。

現在のこのようなコンテナネットワークでは、(1) コンテナ内部の TCP/IP スタック処理、(2) ホスト OS における NAT ルータ処理がオーバーヘッドとなり、物理ホストと比較してコンテナ環境のネットワーク性能が大きく低下する要因となっている。この性能低下を確認するため予備実験では、40Gbps のリンクで接続された 2 台の物理ホスト間で、片方のホストまたはホスト上のコンテナから、対向ホストへ TCP でトラフィックを送信した場合のスループットと遅延を計測した。図 2 は iperf3 を用いたスループットの、図 3 は sockperf を用いた遅延の計測結果を示している。図 2 から、NAT を経由するコンテナのスループットは物理ホストの半分以下である約 18Gbps となった。また遅延もコンテナでは物理ホストに対して約 1.3 倍と、コンテナ環境での大幅な性能低下が確認できる。このように、コンテナ技術は特立した実行環境をより多く集約できる一方で、帯域と遅延の両面において性能低下がある。

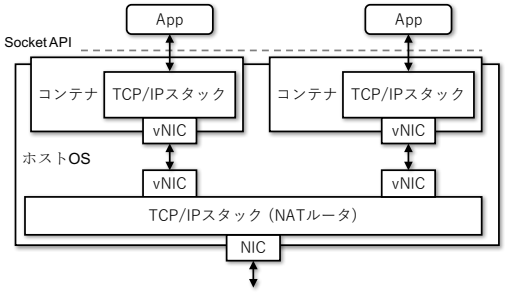


図 1 NAT によるコンテナネットワーク

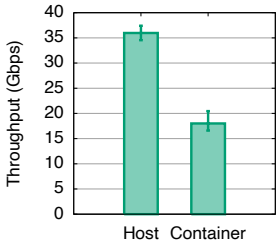


図 2 スループット

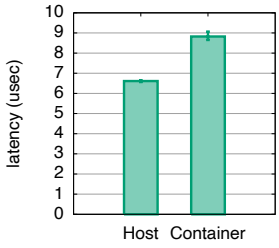


図 3 遅延

２．研究の目的

本研究では、前節で述べたコンテナネットワークにおける性能低下を回避し、コンテナ仮想化環境においても物理ホストと同じだけのネットワーク性能を実現することを目的とする。コンテナネットワークの性能を向上することによって、マイクロサービスアーキテクチャやサーバレスコンピューティングなど、コンテナによるアプリケーション実行を前提とする新しいコンピューティング基盤において、ネットワークのボトルネック化を回避し、仮想基盤がより効率的に物理ホストの資源を利用できるようになる。

３．研究の方法

本研究では、図 1 示すように、コンテナネットワークにおけるデータパスが二度 TCP/IP スタックを経由することに着目し、プロトコル処理を減らすことでネットワークを高速化する手法を提案する。具体的には、コンテナ上のアプリケーションが、コンテナによる分離を越えてホスト OS の TCP/IP スタックを直接利用するアーキテクチャを提案する。提案手法である Socket Grafting は、図 4 に示すように、コンテナ内部のソケット層とホスト OS のソケット層を繋ぐ新しいデータ通信チャネルである。Socket Grafting では、コンテナ内部のアプリケーションがソケットに書き込んだデータはそのままホスト OS へ運ばれ、ホスト OS のネットワークスタックに作成されたソケットに書き込まれる。これによって、コンテナ上のアプリケーションからデータを送信する際に通過するネットワークスタックがひとつになり、物理ホストと同等のネットワーク性能を得ることができる。

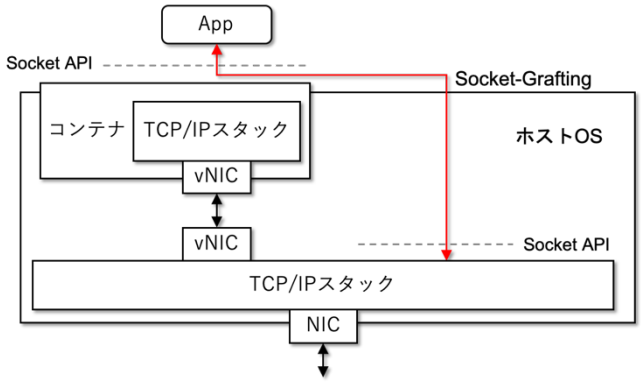


図 4: Socket-Grafting によるデータパス

本研究の大きな利点の一つは、既存の汎用 OS の TCP/IP スタックをそのまま利用できる点である。既存の高速パケット I/O を用いたネットワークスタックの再実装というアプローチは、Linux をはじめとする汎用 OS が実装してきた様々な機能を使うことができないという欠点がある。一方で Socket Grafting は、TCP/IP プロトコルスタックの上位、ソケット層に新しいデータ通信チャネルを構成するものであり、TCP/IP プロトコルスタックの実装に依存しない。そのため、汎用 OS の成熟した TCP/IP スタックが持つ様々な機能をそのまま利用できると同時に、オープンソースコミュニティの継続的な開発の恩恵を受けることができる。結果的に、実運用の観点から、本提案手法は現実のシステムへの提供可能性が高い手法であると言える。

４．研究成果

Socket Grafting の具体的な実装として、AF_GRAFT と呼ばれる新しいソケットアドレスファミリを Linux に実装した。AF_GRAFT ソケットは、bind()システムコールを通じて別のソケットに接続される。この接続は、Linux でコンテナごとにネットワーク空間を分離する Network namespace を越えて行うことができる。これによって、コンテナ上のアプリケーションは、AF_GRAFT ソケットを通じてホスト OS のソケットを利用することができる。下記から、AF_GRAFT ソケットの API について述べる。

```
struct sockaddr_gr {
    __kernel_sa_family_t sgr_family;
    char sgr_epname[AF_GRAFT_EPNAME_MAX];
};
```

AF_GRAFT は接続する先のソケットを bind()システムコールを通じて指定するため、新しいソケットアドレスファミリ構造体を導入した。上記の sockaddr_gr 構造体は、アドレスファミリの識別子に続いて、接続先のソケットを示す文字列を持つ。ある文字列がどのようなソケットにマッピングされるかは、Linux のネットワークスタックの制御コマンドである iproute2 を拡張した下記のコマンドで設定される。この例では、"ep-http"という文字列は、ホスト OS の IPv4 アドレス 10.0.0.2、ポート番号 80 番に開いたソケットに接続されることを示している。また、このマッピングをコンテナごとに制御することによって、コンテナごとに利用できるホスト OS のネットワーク資源を分離し、アクセス制御することが可能である。

```
$ ip graft add ep-http type ipv4 addr 10.0.0.2 port 80
$ ip graft add ep-out type ipv4 addr 10.0.0.2 port any
$ ip graft del ep-un
```

マッピングを事前に設定し、アプリケーションは AF_GRAFT ソケットに sockaddr_gr 構造体を bind することで、通常のソケットと同様に AF_GRAFT ソケットを利用することができる。

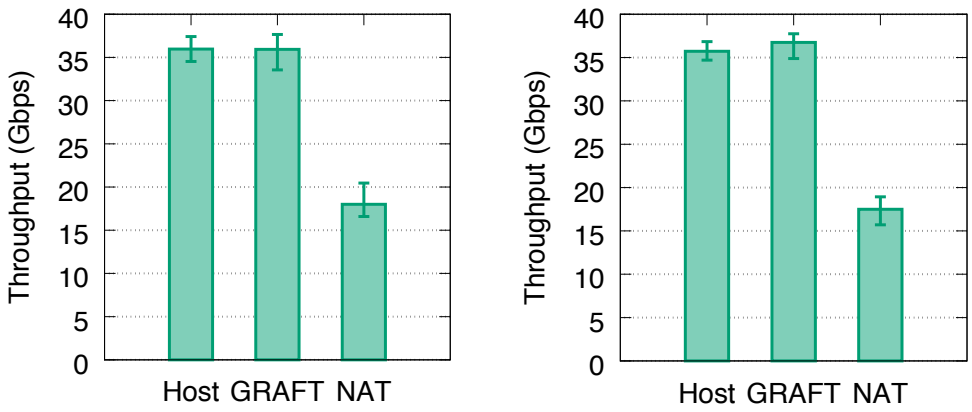
```
int sock;
struct sockaddr_gr sgr;

sock = socket(AF_GRAFT, SOCK_STREAM, 0);

sgr.sgr_family = AF_GRAFT;
strncpy(sgr.sgr_epname, "ep-test", 7);
bind(sock, (struct sockaddr *)&sgr, sizeof(sgr));
```

AF_GRAFT は新しいアドレスファミリであるため、既存のアプリケーションで利用するためにはソースコードの変更が必要である。この問題を回避してより簡単に AF_GRAFT を利用するため、LD_PRELOAD を利用したハイジャックライブラリの実装も行った。

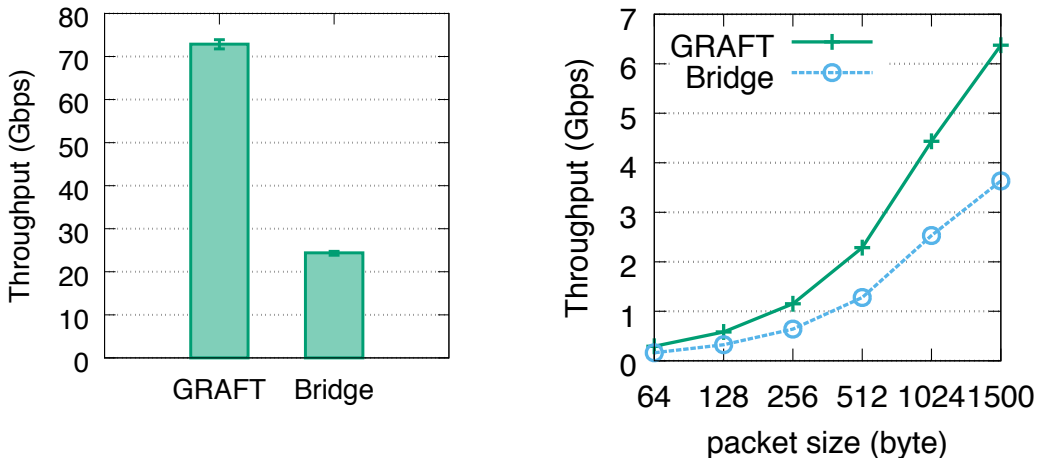
図 5 に、AF_GRAFT を用いてコンテナ環境におけるネットワーク性能を計測した結果を示す。実験では、Intel Core i7-3770K CPU、32GB DDR3 メモリ、Mellanox ConenctX-4 LX 40Gbps を搭載したマシンを 2 台用意して 40Gbps で接続し、片方のマシンのコンテナ上でベンチマークアプリケーション(スループットの試験では iperf3、遅延の試験では sockperf)を動作させ、対向の物理マシンでその性能を計測した。図中 Host は物理ホストでの性能を、GRAFT はコンテナ環境で AF_GRAFT を用いた場合の性能を、NAT はコンテナ環境で一般的な NAT によるネットワーク構成を用いた場合の性能をそれぞれ示している。本実験の結果から、AF_GRAFT はコンテナ環境においても物理ホストと変わらない性能を実現できることが示された



(a) 送信　　(b)受信

図 5 1 TCP フローによるスループット計測

図 6 は、1 台の物理ホスト上に起動した 2 台のコンテナ間でスループットを計測した結果を示している。同一ホスト上のコンテナ間通信の場合、通常のコンテナネットワーク構成(図中 Bridge)ではパケットがホスト OS のソフトウェアブリッジを経由しなければならないのに対して、AF_GRAFT による通信ではループバックインターフェース経由の通信になる。ループバックインターフェースは MTU サイズが大きいため、TCP による計測では大きな性能向上を得られた。一方 UDP による計測では MTU サイズに起因する性能向上は無いものの、コンテナ内部のネットワークスタックのバイパスによる性能の向上が見られた。



(a) TCP トラフィック　　(b) UDP トラフィック

図 6: 同一ホスト上の 2 台のコンテナによるスループット計測

本研究の成果は、国際会議での発表と Best Paper Award の受賞に加えて、Linux のネットワークスタック開発者の会議である Linux NetDev Conference でも発表を行い、Linux カーネル開発者らと本手法の有効性について議論を行った。また実装した AF_GRAFT のソースコードはオープンソースとして公開している。

５．主な発表論文等

〔雑誌論文〕　計2件（うち査読付論文　2件／うち国際共著　0件／うちオープンアクセス　1件）

１．著者名 Nakamura Ryo、Sekiya Yuji、Tazaki Hajime	４．巻
２．論文標題 Grafting sockets for fast container networking	５．発行年 2018年
３．雑誌名 ANCS '18 Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems	６．最初と最後の頁 15-27
掲載論文のDOI（デジタルオブジェクト識別子） 10.1145/3230718.3230723	査読の有無 有
オープンアクセス オープンアクセスとしている（また、その予定である）	国際共著 －

１．著者名 Nakamura Ryo、Kuga Yohei、Sekiya Yuji	４．巻
２．論文標題 An Alternative Fast Packet I/O with Native System Calls	５．発行年 2019年
３．雑誌名 CFI'19: Proceedings of the 14th International Conference on Future Internet Technologies	６．最初と最後の頁 1-7
掲載論文のDOI（デジタルオブジェクト識別子） 10.1145/3341188.3341193	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 －

〔学会発表〕　計1件（うち招待講演　0件／うち国際学会　1件）

１．発表者名 Ryo Nakamura
２．発表標題 AF_GRAFT: A new address family for containerized applications
３．学会等名 The Technical Conference on Linux Networking (Netdev 0x13)（国際学会）
４．発表年 2019年

〔図書〕　計0件

〔産業財産権〕

〔その他〕

－

６．研究組織

	氏名 （ローマ字氏名） （研究者番号）	所属研究機関・部局・職 （機関番号）	備考