Vol.2019-IOT-44 No.21
2019/3/7

分散型データセンターOSを目指したリアクティブ性を持つコンテナ実行基盤技術

松本 亮介[1,a)]　坪内 佑樹[1]　宮下 剛輔[1]

概要：コンピュータリソースを集約し，リソース効率化やコスト削減を行うために，これまで大規模データセンターが建設されてきた．一方，データセンターやクラウドサービスの利用者の観点で，インターネット上で各種サービスを提供する企業では，システムの安定性と耐障害性を高めるために，クラウドネイティブやマルチクラウドに基づいたマイクロサービスアーキテクチャが注目を浴びている．マイクロサービスアーキテクチャでは，全体のサービスを細かいサービスに分割して抽象化，かつ，疎結合な状態にして再利用性を高め，採用するソフトウェアやベンダーの変化に対して強いシステムを構築する．一方で，マイクロサービス間での通信を高速に行い，高品質なサービスを提供するために，いかにレイテンシーを低減させるかが課題になってきている．また，データセンター事業者においても，巨大なデータセンターを建設する際に，この技術変化の速い時代において，適切な収容の見積もりを行うことは困難である．本研究では，コンテナ型データセンターだけでなく，より小規模なデータセンターの分散化が進むことを想定して，分散型データセンターを抽象化するための OS の要件と必要な要素技術について検討する．さらに，抽象化されたデータセンター上にプロセスかのように展開されるコンテナに必要な特性を議論した上で，Amazon や Google をはじめとした各社のコンテナ実行基盤の取り組みについても整理して考察する．

Reactive Container Execution Platform Technology Toward Distributed Data Center OS

Ryosuke Matsumoto[1,a)]　Yuki Tsubouchi[1]　Gosuke Miyashita[1]

1. はじめに

大規模なデータセンターを建設し，ハードウェアリソースを集約することにより，コストの削減を行うといったような，Google がデータセンターを立てることのコストメリットや効率化についての研究が過去に報告されてきた [4], [5], [9]．アプリケーションの高度化に伴い，システムの安定性や耐障害性を高めるためにクラウドサービスを利用する機会が増え，マルチクラウド，クラウドネイティブ [3], [35] といったシステム設計手法が採用される時代になってきている．さらに，以前からクラウドサービスを活用している Web サービス企業においてはマイクロサービスアーキテクチャ [13] が注目を浴びている．

マイクロサービスアーキテクチャは，全体のサービスを細かいサービスに分割して抽象化，かつ，疎結合な状態にして再利用性を高め，マルチクラウドと組み合わせることにより，採用するソフトウェアやベンダーの変化に対して強いシステムを構築する．一方で，そのようなシステムアーキテクチャは，全体のサービスを複数のマイクロサービスで抽象化し連携するため，これまで以上にマイクロサービス間での通信が必要になる．そのように，サービス間の通信を高速に行い，サービス全体の品質を向上させたいデータセンターやクラウドサービス利用者にとっては，データセンターまで，あるいは，データセンター間のレイテンシーが課題になってきている．また，データセンターおよびクラウドサービス事業者としては，巨大なデータセンターを建設したあとに，技術変化のはやい時代に，どの

[1] 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 Research Center, SAKURA Internet Inc., Akasaka, Chuo-ku, Fukuoka 810-0042 Japan
[a)] r-matsumoto@sakura.ad.jp

ようにデータセンターやクラウドサービスの利用者を増やしていくのかといった見積もりの難しさもある．

システム設計がクラウドの特性に最適化され，各クラウドベンダーに依存しない形で設計するマルチクラウドやクラウドネイティブ化が進み，クラウドの活用の幅が広がるに応じて，スマートハウスやスマートモビリティ，エッジコンピューティング，データ流通といったように，リアルタイム性が要求される領域がクラウドの活用を検討し始めている [33], [34]．ディザスタリカバリに関する研究でも，災害が起きたときにいかにネットワークの経路やデータサイズを判断して，自動的かつ迅速に安全なデータセンターにデータを移動させるといった話題 [20], [32] も増えてきている．このように，リアルタイム性の課題が解決してクラウドがもっと当たり前に使われていくと，クラウドプロバイダーにとっては，これまで以上に予測できない大量のアクセスやトラフィックが集中することにも繋がる．

今後，データセンター間のレイテンシーの課題を解決することと，予測できない大量のトラフィックに耐えうるデータセンターの構成を取り，コンピュータリソースを適切に管理することが重要である．我々は，こうした要件と時代背景を考慮して，コンテナ型データセンターだけではなく，より小規模のデータセンター，あるいは，データセンターとは呼べないまでも小型のラック群があらゆる場所に分散していく時代になっていくと考えている．データセンターを分散化して様々な場所に建設することによって，データセンター間やエンドユーザーからアプリケーション起動場所までの物理距離を短くし，レイテンシーを少なくする．データセンターの分散と集約，さらには，あらゆる要求においてクラウドが使われる時代になると，どのデータセンターにデータセンターやクラウドサービス利用者のアプリケーション実行環境が起動しているかは抽象化されて隠蔽され，システムの状態に応じて自律的に再構成することが求められる．そのようにデータセンターの集約から分散が進むと，分散されたデータセンターを抽象化するOS の層，及び，実行環境の高集積化やセキュリティ，性能，リソース管理，運用技術といった課題を解決するための基盤技術がソフトウェアとして必要になる．その OS の層を，本論文では分散型データセンター OS と呼ぶ．

我々が提案する分散型データセンター OS の役割は，いつ誰がどこでクラウドを使っても，それがどのデータセンターで動いているかは意識する必要がなく，負荷分散やディザスタリカバリにおいても，反応的かつ自律的にデータセンター間を移動して最適な実行環境をデータセンターやクラウドサービス利用者に提供することにある．マルチクラウドやクラウドネイティブの時代に，データセンターの分散化を進めながら，データセンターの集約に遜色ないリソースの効率化を実現する必要がある．予測できないような突発的なアクセス過多や災害，各種イベントに応じて，アプリケーション実行基盤が分散型データセンター間を反応的かつ自律的に移動することのできるマイグレーション技術や状態変化速度の高速化技術の研究開発によって，突発的なアクセスに対して反応的にリソースを割り当てられる．そのことで，これまでのように，予測できないアクセスに対してプロアクティブにアプリケーションを大量に起動させておく必要がなく，予測できない変化に強いシステムが設計できる．また，同時に，アクセスの変化とリソース使用量の変化を同期できるため，データセンターのリソース使用量を効率化できる．本論文では，Amazon や Google をはじめとした各社のコンテナ実行基盤のアーキテクチャを体系的に整理し，その上でコンテナ実行基盤に基づく分散型データセンター OS の設計に必要な要件や定義，各種機能について議論する．

本稿の構成を述べる．2 章では，分散型データセンター OS とその関連技術について整理する．3 章では，分散型データセンター OS に必要な要件と機能，および，開発中の実装ついて述べ，4 章でまとめとする．

2. 分散型データセンター OS と関連研究

これまでデータセンターの集約化が進んできた中，Web サービスを取り巻くシステムアーキテクチャは抽象化が進んでいる．例えば，マイクロサービスアーキテクチャ?においては，Web サービスにおける各種コンポーネントを小さなサービスとして抽象化し，複数のサービスを組み合わせて大きなシステムを実現している．マイクロサービスによって抽象化を行いながら，全体のシステムも安定化させるためには，マイクロサービス間でのネットワークのレイテンシが重要になってくる．

マイクロサービスのメリットとして，サービスを抽象化かつ疎結合な状態にすることで再利用性を高めながら，1 つのマイクロサービスに問題が起きても，マイクロサービス単位でのスケーリングや耐障害性を高める構成をとりやすくなる．一方で，マイクロサービスアーキテクチャのメリットを最大限享受するためには，マルチクラウドやクラウドネイティブ化 [3], [35] を進め，1 つのデータセンターや単一のクラウドサービスに依存しない構成をとる必要がある．そのため，データセンター事業者としては，巨大なデータセンターを一箇所に集約するよりも，クラウドネイティブの時代に即した，データセンターの分散化を考慮する必要がある．様々な場所にデータセンターを分散して建設することによって，データセンター間やエンドユーザーからアプリケーション起動場所までの物理距離を短縮し，レイテンシーを改善する．そのことによって，場所にも依存しない，より信頼性の高いマイクロサービスアーキテクチャを採用することが可能になる．

データセンターの分散と集約，さらには，あらゆるシステム要求に対してクラウドサービスが使われる時代になっ

てくると，データセンターやクラウドサービス利用者にとって，どのデータセンターに自分のアプリケーション実行環境が起動しているかは気にせずに利用したい．むしろ，マルチクラウドやクラウドネイティブが進む時代において，データセンターやクラウドサービス利用者がデータセンターの場所を詳細に知る必要があると，利用者にとっては使いづらいサービスになり得る．

データセンターの集約から分散が進み出すと，分散されたデータセンターをうまく抽象化する OS レイヤー，及び，実行環境の高集積化やセキュリティ，性能，リソース管理，運用技術といった課題を解決するための基盤技術のようなものがソフトウェア的に必要になってくると考えられる．それらを本論文では分散型データセンター OS と呼ぶことにする．ここでいう OS とは，複数のデータセンターを抽象化する機能や，抽象化されたデータセンターリソースをプロセスやスレッドのようにみなせるアプリケーション実行環境に割り当てる機能，分散されたデータセンターを透過的にスケジューリングする機能などを意味する．

例えば，Mesos Marathon[22] ははやくからこの考えのもとに様々な議論や実装を進めている．また，松本らは，Web サーバの高集積マルチテナントアーキテクチャに関する研究 [36] を行っており，単一の収容サーバに高集積にホストを収容しながら，いかにセキュリティや性能，リソース管理，運用技術を向上させるかという観点で類似している．本章では，よりコンテナの実行環境や現在の時代背景に合わせて，関連技術を整理する．

分散型データセンター OS の役割は，いつ誰がどこでクラウドを使っても，それがどのデータセンターで動いているかは意識する必要がない．さらに，負荷分散やディザスタリカバリにおいても，できるだけ自動的にデータセンター間を移動して最適な実行環境をデータセンターやクラウドサービス利用者に提供することにある．ディザスタリカバリに関する研究でも，災害が起きたときにネットワークの経路 [32] やデータサイズ [20] を判断して，自動的に安全なデータセンターにデータを移動させる手法が提案されている．この分野においても，レイテンシーや分散と集約の課題は重なる部分がある，

このように，マルチクラウドやクラウドネイティブの時代に，分散型データセンターを実現するためには，分散型データセンターを薄く繋いだレイヤーを提供する分散型データセンター OS を検討することが必要である．さらに，予測できないような突発的なアクセス過多や災害，イベントに応じて分散型データセンターを自由に移動できるマイグレーションや状態変化の高速化といった，リアクティブ性を持った実行基盤技術の研究開発が必要になる．

Google や Amazon など，データセンターを持つクラウドベンダーは，各社コンテナ実行基盤の基盤技術を開発している．以下では，runC[6]，Firecracker[12]，gVisor[16]，Nabla-Containers[25]，Kata-Containers[18] を例にその基盤技術について整理する，

2.1 コンテナ実行環境の基盤技術

コンテナ実行環境とは，kubernetes[31] を中心とするコンテナオーケストレーションに必要な機能のうち，Open Container Initiative (OCI) の Runtime Specification v1.0.0 と Image Format Specification v1.0.0[27] に基づいて起動する，コンテナプロセスが実行される環境を意味する．kubernets と連携してコンテナを実行する機能をコンテナランタイムと呼ぶが，コンテナランタイムは大きく分けて CRI ランタイム (High-level container runtime) と OCI ランタイム (Low-level container runtime) に区別できる．CRI ランタイムはオーケストレーション層からのコンテナに関するパラメータを取得し，Pod と呼ばれるコンテナの起動スペースの管理や，コンテナのイメージ管理，その他コンテナの起動に必要な情報を OCI ランタイムに渡して指示する機能等を提供する．OCI ランタイムは CRI ランタイムから受け取った情報に基づいて，ランタイム専用の Pod の起動処理や，コンテナプロセスを適切な Pod 上に起動させる機能等を提供する．

コンテナプロセスは，コンテナが起動するホスト OS 上のカーネルを共有し，chroot() システムコールや unshare() システムコール等，様々なプロセスの隔離技術を組み合わせることで隔離環境を実現する．そのようなコンテナプロセスをホスト OS 上に起動する際に，複数のコンテナプロセスはホスト OS を共有するために，各コンテナプロセス間でのセキュリティやリソース管理，パフォーマンスを考慮する必要がある．

コンテナ実行環境の基盤技術は現状以下の 5 つに分類される．

(1) プロセス
(2) サンドボックス
(3) ユニカーネル
(4) microVM
(5) VM

(1) のプロセスアプローチは，ホスト OS 上に cgroup() や unshre() システムコールによって隔離した Pod を作成し，Pod 内にコンテナを起動させる．これによって，Pod 内で起動する単一，あるいは，複数のコンテナは，Pod で隔離されたリソースやネームスペース，ネットワークを共有し，さらにコンテナ単位で隔離を行う．そのため，各 Pod やコンテナは，ホスト OS のカーネルを共有することになるため，カーネルに異常が生じた場合は影響を受ける．OCI が Runtime specification を準拠した実装として公開している runC がこのアプローチに含まれる．従来のコンテナのプロセス隔離の手法を利用しているため，Pod やコンテナを簡単に起動できて，状態の変化も高速である．一

方で，プロセス単位での隔離であり，ファイルシステムやシステムコール等の精緻な制限はしていないため，OS の脆弱性の影響を受けやすい．

(2) のサンドボックスアプローチは，(1) のアプローチと同様，コンテナの起動スペースである Pod やコンテナが複数起動した場合，Pod，あるいは，コンテナ間でホスト OS のカーネルを共有する．一方で，サンドボックスアプローチのコンテナは，ユーザー空間上に仮想的に最低限のカーネル環境を構築して，コンテナ上のアプリケーションのアクセス制御を行う．その上で，kubernets の Pod の仕様に基づき，ユーザー空間上に作成された仮想カーネル環境単位で Pod が構成される．例えば，サンドボックスアプローチにおける代表的な実装である gVisor は，コンテナ上で起動するアプリケーションのシステムコールを sentry とよばれるユーザー空間上のカーネルが ptrace[28] によって監視する．さらに sentry は seccomp によって，ptrace で監視しているシステムコールを制限する．ファイルシステムのアクセスについては，sentry とは別に gofer と呼ばれるプロセスによってアクセス制御を行う．以上の特徴から，Pod の起動やコンテナプロセスの起動は，ユーザー空間上で行われるため，高速に起動できるという特徴がある．一方で，ユーザー空間上のカーネルやホスト OS のカーネルに異常が生じたり，脆弱性があった場合は，複数のコンテナが影響を受ける可能性がある．また，ユーザー空間上でのアクセス制御の影響から，アプリケーションのパフォーマンスや I/O，ネットワーク性能がオーバーヘッドになるという報告もある [19]．

(3) のユニカーネル [21] アプローチは，ハイパーバイザ上に起動可能なユニカーネル OS の HW 仮想化領域を，ユーザ空間におけるコンテナによる隔離技術に置き換えて起動させる．ユニカーネルは本来，単一のアプリケーション専用のカーネルを用意し，起動させるアプローチをとるため，アプリケーション専用の最適化をカーネルに適用できるというメリットがある．一方で，アプリケーションとカーネルレイヤー間の抽象化や実装が複雑になったり，適切に動作するアプリケーションがユニカーネルの仕様に強く依存する．例えば，Nabla-Containers は，solo5 のユニカーネル実装を採用しており，ユニカーネルの HW 仮想化領域をコンテナによる隔離技術に置き換えることで，Linux ホスト OS 上のユーザー領域にユニカーネルを展開させる．そして，solo5 用にビルドされたアプリケーションイメージを使って，アプリケーションをユニカーネル上に起動させる．Pod も Nable-Containers 専用の Pod を採用し，ユニカーネルである solo5 単位で Pod を作成し，Pod 内でコンテナを起動させる．そのため，Pod もコンテナの起動もユーザー領域のカーネルの起動となるため高速に起動する．また，ユニカーネルとホスト OS の境界では，システムコールを 7 個まで制限しており，Horizontal Attack Profile の観点で，他のランタイムよりもセキュアであるという方向もある [2]．

(4) の microVM[26] アプローチは，ホスト OS 上に，セキュリティや状態変化の効率性を最大化するために，コンテナが起動に必要な最低限のカーネルの 4 つの機能（virtio-net，virtio-block，serial console，a 1-button keyboard controller）だけをロードした軽量 VM 上にコンテナを起動させる．そのため，Pod は高速かつ軽量に起動する microVM 上で区別され，コンテナは microVM 上に起動する．コンテナ，あるいは，Pod 単位で microVM を起動する必要があるが，コンテナの起動に必要な最低限のカーネルの機能のみをロードするため，VM でありながら状態の変更が高速であるという特徴がある．例えば Firecracker は，microVM アプローチによるハイパーバイザを実現しており，microVM 単位で Pod を構成し，Pod 内では OCI により実装され広く使われている runc によってコンテナを起動する．Firecracker は snapshotter と runtime で構成されている．runtime によって，microVM を管理する VMM と microVM 内の agent が連携して runc を操作し，Pod やコンテナを制御する．また，snapshotter は microVM で動作するコンテナイメージを作成する．snapshotter はブロックデバイスとしてイメージファイルを作成し，イメージファイルを介して microVM にパススルーする．Firecracker の VMM は，REST でリモートから操作可能になっている．

(5) の VM アプローチは，Pod を VM 単位で構成する．そのため，Pod を起動させるためには，VM を起動させる必要があり，起動に時間がかかる．一方で，OpenStack を始めとする従来の VM オーケストレーションや VM の資産を利用することができるため，OS レイヤでの専用の修正が必要なく，シンプルで利用しやすい．また，Pod 単位で VM が分離されるため，従来の VM の隔離程度のセキュリティを容易に確保できる．これまでのハイパーバイザや VM の資産を活用できるため，Pod さえ起動してしまえば，他のアプローチと比べても CPU やメモリ，ネットワーク性能は非常に高い [19]．Kata-Containers はまさにこのアプローチを採用している．また，Kata-Containers で扱う Pod は VM であれば良いので，(4) アプローチの Firecracker の VMM と連携して Pod を起動するという実装も報告されている [10]．

Windows でも従来からピコプロセス [23], [24] という仕組みがある．ピコプロセスは，microVM ほどカーネルレベルでの分離はしないが，ユニカーネルアプローチのようにライブラリ OS を採用し，専用のアプリケーションイメージを実行しながらシステムコールを監視して，そのシステムコールによって Windows カーネルと共存している Linux カーネルの処理をユーザランドに展開し，Linux のプロセスを立ち上げられる．

2.2 リアクティブ性の高いコンテナ実行環境の必要性

これまでは，Web アプリケーションは事前にプロセスやインスタンスを起動させておき，大量のアクセスが想定される場合は予め複数台のインスタンスを起動させておく方式が広くとられていた．しかし，多種多様な Web サービスやスマートフォンを始めとする大量のクライアントが人々の生活に強く依存し，SNS が広く普及したことから，事業者は大量のアクセスや変化を予測することが困難になっている．そこで，クライアントからの SMTP 接続や HTTP リクエストのようなセッション契機でリアクティブにコンテナの状態を変化させることにより，収容サーバに高集積にコンテナを収容しながらも，突発的なアクセスに対して適応的にスケーリングする研究がされている [37]．また，各種クラウドベンダーではサーバレスアーキテクチャに基づいて，AWS lambda[1] をはじめとする各種サービスが提供されている．

ホスト OS と Pod，コンテナ，および，それらを制御するオーケストレーション層は，分散型データセンターを覆うかのような OS とみなせる．そのような分散型データセンター OS の観点で，よりリソースの効率性や状態変化のリアクティブ性を実現するためには，カーネルを共有してサンドボックス環境でコンテナ実行環境を起動させる方法をとればよい．(2) のサンドボックスアプローチは，システムコールレベルでの脆弱性があった場合に，コンテナ間のプロセスで相互に影響を受けない．一方で，(4) の microVM アプローチと違いカーネルは基本的に共有のため，コンテナが動作しているカーネル本体が停止すると，そのカーネルで同様に動いているコンテナは影響を受ける．まさにこれは，データセンター OS の視点から，OS の概念でいうメモリを共有するという意味での軽量スレッドであるといえる．また，(3) のユニカーネルアプローチのように，ホスト OS のカーネルを共有しながら，(2) のサンドボックスアプローチのアクセス制御よりも，従来のユニカーネルの資産を利用して，ユニカーネル専用のアプリケーションイメージを採用し，より厳密なアクセス制御を行うアプローチはスレッドとみなせる．

microVM アプローチに分類される Firecracker では，データセンターにある大量のサーバ上に薄い分散型データセンター OS のようなレイヤーを作り，まるでデータセンター上に大量の軽量 VM，つまり，OS の概念でいくと軽量プロセスが起動するような状態を実現することができる．各軽量 VM は REST でコントロール可能であり，かつ，軽量 VM は最低限のカーネルの機能しか起動時にロードしないので，AWS の報告 [11] によると，125ms 程度で高速に起動できることから，イベントに応じてリアクティブに状態を変えられる．一方で，コンテナランタイム環境が高速に起動できたとしても，コンテナ上で起動する Web サーバ等のミドルウェアが高速に起動できないという課題もある．松本らは，コンテナ上で起動するサーバプロセスを，予め起動完了直後で CRIU[8] によってイメージ化しておくことにより，Apache httpd で起動を 900msec 程度高速化することに成功している [38]．まさに microVM アプローとは，分散データセンター OS における軽量プロセスとみなせる．

(1)(2)(3)(4) のアプローチは，同時にデータセンターやサーバに対する収容効率も大幅に上げることができる．コンテナの起動をはじめとした状態を高速に変更できることは，イベントが生じたときだけ起動すればよく，イベントがない状態では停止状態が許容されることを意味する．つまり，同時セッション数の数だけ起動しておけばよく，これまでのプロアクティブな VM やプロセスの起動で，予め収容数の数だけ起動させておく必要があるのとは，効率性の観点で大きな違いが生じる．このように，データセンターの集約相当のハードウェアコストのメリットを活かすために，ソフトウェアの観点からデータセンターが分散してもコンテナの高集積化を行うことでハードウェアコストのメリットを高めてくというアプローチにもなる．

3. 分散型データセンター OS に必要な要件

マルチクラウドやクラウドネイティブ，さらには，スマートハウスやスマートモビリティ，エッジコンピューティング，データ流通といったように，リアルタイム性の課題，さらには社会にクラウドリソースを適切に活用することが求められている．そのために，クラウド事業者としては，分散型データセンター OS とリアクティブ性を持つコンテナ実行基盤技術に取り組んでいく必要がある．

分散型データセンター OS を設計する中で，2 節でまとめた関連研究に基づき，分散型データセンター OS 上のコンテナ実行基盤に必要な要件をまとめる．

(1) 分散型データセンター OS のプロセスやスレッドとしてのコンテナの概念を定義する
(2) 分散型データセンター OS におけるプロセスやスレッドとしてのコンテナは状況に応じてリアクティブに起動可能にする
(3) 分散型データセンター OS のコンテナを透過的に管理するツール群を用意する

要件 (1) について，2 節の整理に基づくと，コンテナ実行環境における microVM アプローチを分散型データセンター OS における軽量プロセス，ユニカーネルアプローチをスレッド，サンドボックスアプローチを軽量スレッドとみなすのであれば，コンテナの実行環境を単一の VM 上に起動させる VM アプローチは，分散型データセンター OS 上にカーネルを共有しないメモリ空間にコンテナを起動させるという意味でプロセスとみなせる．また，分散型データセンター OS 上における uid/gid など，認証や権限の考え方についても検討する必要があるが，本論文では言及し

ない．

要件 (2) について，分散型データセンター OS を考える場合には，OS における概念と同様に，プロセス型やスレッド型，あるいは，軽量スレッド型のコンテナ実行基盤を，必要に応じて選択し利用できるようにする必要がある．また，全てのコンテナ実行環境は，アプリケーションの実行に応じてリアクティブに起動可能にし，かつ，一定時間起動した後やアプリケーションが起動しておく必要がない場合は停止できるようにする．このような状態変化を高速にすることで，突発的なアクセス集中に対してリアクティブにスケーリングを行えるようにする．それによって，プロアクティブにアプリケーションを大量に起動させる必要がなくなり，アクセスの変化とリソース使用量の変化をできる限り同期できるため，リソース使用量を効率化できる．また，予測できないアクセス集中に対しても，反応的にアプリケーション実行基盤に対してリソースを割り当てられるため，予測できない変化にも強い基盤になる．さらに，停止状態を許容することによって収容効率を上げ，分散型データセンターでありながら，集約型データセンターのリソース効率化に遜色ない状態を担保すべきである．

要件 (3) について，分散型データセンター OS 上に展開されるプロセスやスレッドとしてのコンテナを，分散型データセンター OS 管理者が適切に管理しなければならない．分散型データセンター OS は，データセンターの分散化が進み，透過的な層としてのホスト OS 上に展開されるため，ネットワークを介して統合的にプロセスやスレッドの状態を把握する必要がある．

3.1 k2i:分散型データセンター OS のプロセス情報取得

分散型データセンター OS が進んだ際には，各ホスト OS のローカル環境で ps や top コマンドを実行するのではなく，データセンター OS 全体に対してそれらのコマンドに相当する処理を実現する必要がある．各 OS 上に k2i を起動してプロセスやスレッドの情報を監視し，その情報を REST 経由で回収し，分散型データセンター OS 単位でプロセスの状態を取得，解析できるようにするミドルウェアである．

特定のサービス基盤に対して consul[7] を使って複数のサーバにコマンドを発行する手法もあるが，k2i はプロセス情報を収集することに特化し，各種コマンドやミドルウェア，さらには OS の管理プロセスなどから情報を取得できる使い方を想定して開発している．例えば，Firecracker は VMM に対して REST 経由で命令を送ることにより，microVM の操作を可能にしている．k2i においても，同様に REST 経由でプロセスのリソース情報や操作を可能にする．

UNIX や Linux における ps や top をはじめとしたプロセスの各種管理コマンドを，分散型データセンター OS に対応させたコマンドとして，k2i を利用して開発できるようにする．

3.2 middlecon:分散型データセンター OS のプロセス間通信

これまで実行環境の状態変化に対してリアクティブ性というものがそこまで求められなかったため，システム連携で利用される MQTT や AMQP をはじめとするキューの処理 [17] は非同期で処理することが選択されてきた．一方で，データセンター OS においては，予測できないような突発的なアクセス過多や災害，イベントに応じて大量のホスト OS や Pod で構成される分散型データセンターを透過的に移動することのできるマイグレーションといった，リアクティブ性を持った実行基盤技術が必要になる．

そこで，middlecon は，ミドルウェア，コンテナ実行環境，コンテナランタイム，あるいは，OS といったソフトウェアが，多数のプロセスに対して高速かつ相互に同期的にメッセージを交換できる．OS の概念においては，プロセス間通信を意味する．middlecon は，マイクロサービスアーキテクチャで利用される HTTP/2 や gRPC[15] とは違い，常時多数のプロセス間と TCP セッションを維持しながら，多数のプロセスに対してメッセージを交換できる，プロセスのチャットのような動作する．その機能を利用して，プロセス間でのリソース情報や状態を共有し，イベントに応じて素早く同期的に多数のプロセスにメッセージを送信することにより，反応的かつ自律的な状態変化のイベントを多数のプロセスと共有できる．k2i と同様に，middlecon をホスト OS 上に起動させておくことで，コンテナ実行環境間などで高速かつ同期的にメッセージ通信を行えるようになり，変化に対するリアクティブ性を担保できる．middlecon は高速に多数のプロセスと同期的にメッセージ交換を行えるように，高速な処理とプロトコルを目指している．

MQTT といったメッセージングソフトウェアの前提との違いとして，レイテンシーの課題や小型デバイスのための非同期前提の仕組みと違い，データセンターの分散化が進みレイテンシーが短くなった比較的信頼性のあるネットワークの状況において，計算機リソースを十分に持つプロセスは同期的かつリアクティブに処理できる機会が増えることが挙げられる．例えば，オートスケーリングや予測できない突発的なイベントに反応的な処理を行う状況が挙げられる．実行環境やアプリケーションの状態変化の速度が遅い状況においては，非同期メッセージングが必要であるが，その状態変化が高速になるに従って，リアクティブかつ同期的にプロセスやスレッドを操作できることが，分散型データセンター OS のリソース効率化を向上させる．

4. まとめ

本論文では，コンテナの実行環境の普及に伴って，クラウドネイティブやマルチクラウドのようなシステム構築が進む中，データセンターが集約から分散に向かうであろうことを述べた．データセンターの分散化に応じて，データセンターや OS のリソース管理や高集積にサーバやコンテナを収容することの重要性，セキュリティ，パフォーマンス，運用技術について議論し，分散型データセンター OS の必要性について言及した．既に各種様々なコンテナ実行環境が実装・公開される中で，コンテナ実行環境のアーキテクチャを整理し分類した上で，分散型データセンター OS に必要な要件を検討した．

今後は，分散型データセンター OS に必要な要件を更に深く議論し，要件に応じた OS の設計と実装を行っていく．

参考文献

[1] Amazon Web Services: Lambda, https://aws.amazon.com/lambda/.
[2] Aravena R, Bottomley J, Container Security & Multi-Tenancy Tales from Kata & Nabla, KubeCon CloudNativeCon North America 2018, https://schd.ws/hosted_files/kccna18/20/Container%20Security%20and%20Multi-Tenancy%20Tales%20from%20Kata%20and%20Nabla.pdf.
[3] Balalaie A, Heydarnoori A, Jamshidi P, Microservices architecture enables devops: Migration to a cloud-native architecture, IEEE Software, Vol.33, No.3, pp.42-52, 2016.
[4] Barroso L A, Clidaras J, Hölzle U, The datacenter as a computer: An introduction to the design of warehouse-scale machines, Synthesis lectures on computer architecture, Vol.8, No.3, pp.1-154, 2013
[5] Barroso L A, Hölzle U, The case for energy-proportional computing, Computer, Vol.40, No.12, pp.33–37, 2007.
[6] CLI tool for spawning and running containers according to the OCI specification, https://github.com/opencontainers/runc.
[7] Consul, https://www.consul.io/.
[8] CRIU – A project to implement checkpoint/restore functionality for Linux, https://criu.org/.
[9] D Abts, M R Marty, P M Wells, P Klausler, H Liu, Energy proportional datacenter networks, SIGARCH Computer Architecture News, Vol.38, No.3, pp.338-347, June 2010.
[10] Ernst E, Kata Containers 1.5 release, https://medium.com/kata-containers/kata-containers-1-5-release-99acbaf7cf34.
[11] Firecracker – Lightweight Virtualization for Serverless Computing, https://aws.amazon.com/jp/blogs/aws/firecracker-lightweight-virtualization-for-serverless-computing/.
[12] Firecracker: Secure and fast microVMs for serverless computing, https://firecracker-microvm.github.io/.
[13] Fowler M, Lewis J, Microservices. http://martinfowler.com/articles/microservices.html.
[14] Ferdman M, Adileh A, Kocberber O, Volos S, Alisafaee M, Jevdjic D, Falsafi B, Clearing the clouds: a study of emerging scale-out workloads on modern hardware, ACM SIGPLAN Notices, Vol. 47, No. 4, pp. 37-48, March 2012.
[15] gRPC, https://grpc.io/.
[16] gVisor: Container Runtime Sandbox, https://github.com/google/gvisor.
[17] J E Luzuriaga, M Perez, P Boronat, J C Cano, C Calafate, P Manzoni, “A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks,” in 12th Annual IEEE Consumer Communications and Networking Conference, pp.931–936, 2015.
[18] Kata Containers - The speed of containers, the security of VMs, https://katacontainers.io/.
[19] Xu Wang, Kata Containers and gVisor: a Quantitative Comparison - OpenStack, KubeCon CloudNativeCon North America 2018, https://www.openstack.org/assets/presentation-media/kata-containers-and-gvisor-a-quantitave-comparison.pdf.
[20] Ma L, Su W, Wu B, Taleb T, Jiang X, Shiratori N, ε-time early warning data backup in disaster-aware optical inter-connected data center networks, IEEE/OSA Journal of Optical Communications and Networking, Vol.9, No.6, pp.536-545, 2017.
[21] Madhavapeddy A, Mortier R, Rotsos C, Scott D, Singh B, Gazagnaire T, Smith S, Hand S, Crowcroft J, Unikernels: Library operating systems for the cloud, SIGPLAN Not, Vol.48, No.4, pp.461–472, Mar 2013.
[22] Mesosphere, Inc., Marathon: A container orchestration platform for Mesos and DC/OS, https://mesosphere.github.io/marathon/.
[23] J Howell, B Parno, J R Douceur, How to run POSIX apps in a minimal picoprocess. In 2013 USENIX Annual Technical Conference, pp.321–332, June 2013.
[24] A Baumann, M Peinado, G Hunt, Shielding applications from an untrusted cloud with haven. In USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2014.
[25] Nabla containers: a new approach to container isolation, https://nabla-containers.github.io/.
[26] Ogel Frédéric, THOMAS Gaël, FOLLIOT Bertil, Supporting efficient dynamic aspects through reflection and dynamic compilation, Proceedings of the 2005 ACM symposium on Applied computing, pp.1351-1356, 2005.
[27] Open Container Project, The Open Container Initiative, https://www.opencontainers.org/.
[28] Pradeep Padala, Playing with ptrace, Part I, Linux Journal, Vol.2002, Issue.103, p5, 2002.
[29] Roberts M, Serverless Architectures, https://martinfowler.com/articles/serverless.html.
[30] SECure COMPuting with filters, https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt.
[31] The Kubernetes Authors, kubernetes: Production-Grade Container Orchestration, https://kubernetes.io/.
[32] 江戸麻人，和泉諭，阿部亨，菅沼拓夫:災害リスクを考慮したネットワークの経路制御手法の提案と評価，電気学会論文誌 C，Vol.137, No.3, pp.532-541 2017 年.
[33] 総務省, スマート IoT 推進戦略, http://www.soumu.go.jp/main_content/000439133.pdf.
[34] 総務省, 総務省におけるスマートシティの展開について, https://www.kantei.go.jp/jp/singi/tiiki/kokusentoc/supercity/dai2/shiryou2.pdf.
[35] 千葉立寛, クラウドネイティブ時代に振り返るコンテナの

これまでとこれから, 情報処理, Vol.59, No.11, pp.1022-1027, 2018 年.
[36] 松本 亮介, Web サーバの高集積マルチテナントアーキテクチャに関する研究, 京都大学博士 (情報学) 学位論文, https://repository.kulib.kyoto-u.ac.jp/dspace/handle/2433/225954, 2017.
[37] 松本亮介, 近藤宇智朗, 三宅悠介, 力武健次, 栗林健太郎, FastContainer: 実行環境の変化に素早く適応できる恒常性を持つシステムアーキテクチャ, インターネットと運用技術シンポジウム 2017 論文集, 2017, 89-97(2017-11-30), 2017 年 12 月.
[38] 松本亮介, 近藤宇智朗, 栗林健太郎, CRIU を利用した HTTP リクエスト単位でコンテナを再配置できる低コストで高速なスケジューリング手法, 研究報告インターネットと運用技術 (IOT), Vol.2018-IOT-44, pp.1-8, Mar 2019.