Vol.2018-OS-142 No.8
2018/2/28

クラウドにおけるVM内コンテナを用いた低コストで迅速な自動障害復旧

森川 智紀[1]　光来 健一[1]

概要：様々なサービスがクラウド環境へ移行し，仮想マシン（VM）を用いたサービス提供が増加している．それにつれて，VM やそれが動作するホスト，データセンタに障害が発生した際の対策が必要とされるようになってきた．しかし，従来の障害対策では，障害対策のためのコストと復旧にかかる時間がトレードオフになっていた．そこで本研究では，低コストで迅速な自動障害復旧システム VCRecovery を提案する．VCRecovery では，VM の内部でコンテナを用いることにより障害対策のコストを削減し，障害からの復旧時間を短縮することを可能にする．復旧後に性能が不足する場合にはコンテナを別の VM にマイグレーションすることで対処する．VCRecovery を LXD や Zabbix などを用いて実装し，ウォームスタンバイとコールドスタンバイについて実験を行った．その結果，迅速な障害復旧と低コストを両立させられることが確認できた．

1. はじめに

ウェブサービスやデータベースサービスをはじめとした様々なサービスを提供する際に，クラウド環境を活用することが増えている．クラウドへの需要が高まっていることから，VM やそれが動作するホスト，データセンタに障害が発生した際の対策が重要になってきた．例えば，アクティブ・スタンバイ方式を用いる場合には運用系と待機系を用意し，運用系に障害が発生した際に待機系に切り替えることで復旧を行う．

しかし，障害復旧にかかる時間と障害対策のためのコストがトレードオフになり，両方を満足させることは難しかった．例えば，ウォームスタンバイでは，障害発生後に短時間で待機系に切り替えることができるため復旧時間は短いが，待機系で VM を常時動作させておく必要があるためコストが高い．一方，コールドスタンバイでは，障害に備えるためのコストは低いが，待機系での VM の起動に時間を要するため復旧に時間がかかる．

そこで本稿では，VM 内部でコンテナを用いることにより，低コストで迅速に障害復旧を行うことができるシステム VCRecovery を提案する．VCRecovery では待機系において 1 台の VM だけを動かし，運用系と同数のコンテナを用いて VM の内部にサービスを集約する．これにより，迅速な障害復旧と低コストを両立させる．ウォームスタンバイの場合，障害が発生していない間は待機系の VM 1 台分のコストだけで済み，障害発生後はコンテナにサービスを切り替えることで復旧が可能である．また，コールドスタンバイの場合，障害発生時には待機系の共用 VM の中でコンテナを起動することにより，VM を起動する従来手法よりも迅速に復旧を行うことができる．復旧後に VM の負荷が高まった場合はコンテナをより大きな VM へとマイグレーションさせることで性能を改善する．

我々は KVM 上の VM 内部で LXD コンテナ [1] を用いて VCRecovery の実装を行った．LXD はコンテナ型ハイパーバイザと呼ばれ，コンテナのライブマイグレーションが可能である．また，libvirt-snmp[2] と Zabbix[3] を組み合わせることで障害検知から復旧までを自動化した．libvirt-snmp を用いて，VM に監視エージェントを組み込むことなく VM の状態を SNMP で通知する．Zabbix を用いて，障害を検知した際に待機系で復旧用スクリプトを実行する．復旧時にサービス提供元を切り替えるために，ダイナミック DNS[4] を用いた．

VCRecovery を用いて障害復旧時間を測定する実験を行った．VM が異常終了する障害を想定してウォームスタンバイとコールドスタンバイについて実験を行った．ウォームスタンバイにおいてはコンテナを用いても復旧時間は従来と同程度で済むことが分かった．コールドスタンバイにおいては VM よりもコンテナの起動が高速であるため，復旧時間を短縮することができた．一方，VM 内コンテナを用いることによるサービスの性能低下は 6.6%であ

[1] 九州工業大学
Kyushu Institute of Technology

ることが分かった．

以下，2 章でクラウドの障害と対策について述べ，3 章で VM 内コンテナを用いることにより低コストで迅速な自動障害復旧を可能にするシステム VCRecovery を提案する．4 章で VCRecovery の実装について示し，5 章で VCRecovery における復旧時間とコストについて調べた実験について述べる．6 章で関連研究に触れ，7 章で本稿のまとめと今後の課題について述べる．

2. クラウドの障害と対策

クラウドを用いて様々なサービスを提供することが増えており，クラウドに障害が発生した場合，多数のユーザに影響をおよぼす．過去には管理ミスで Amazon S3 が停止するという事例 [5] や停電により Amazon EC2 が停止するという事例 [6] があるように，クラウドにおける障害の発生は常に想定しておかなければならない．そこで，障害発生から復旧までを迅速に，かつ低コストで行える障害対策が求められている．クラウドでは VM やそれが動作するホスト，データセンタなど様々な規模で過負荷による性能低下や接続障害などが起こるため，それぞれの規模に合わせた適切な障害対策・復旧手法を用いる必要がある．

その中で様々な障害対策が考えられているが，本稿では図 1 のように運用系と待機系を用いるアクティブ・スタンバイ方式を考える．障害発生前は運用系でサービス提供 VM を動作させ，障害発生後は待機系で復旧用 VM を動作させる．そして，サービス利用者のアクセスを待機系に切り替えることで復旧を完了する．障害発生から待機系へのサービス切り替えが完了するまでの時間が障害復旧時間となる．一方，クラウドの課金は VM 単位であることが多いため，平常時に復旧用 VM を管理するためにかかるコストが障害対策のコストとなる．

しかし，従来の手法では障害復旧時間と障害対策のコストがトレードオフになっている．例えば，Amazon AWS では障害対策としてウォームスタンバイやコールドスタンバイなどが利用可能である [7]．ウォームスタンバイでは，運用系で動作させている VM と同じ VM を待機系でも動作させておく．そのため，運用系で障害が発生した場合はすぐに待機系の VM に切り替えることができ，短時間で復旧できる利点がある．しかし，障害発生まで使用されない復旧用の VM を常に起動しておき，データベース等の同期を取り続ける必要があるため，コストが高くなる．このコストを抑えるために待機系ではできるだけ小さな VM を動かすようにすることもできるが，その場合には復旧後にサービスを動作させる VM を切り替える必要がある．その際にダウンタイムが生じる可能性がある．

一方，コールドスタンバイでは，運用系で動作させている VM のバックアップをストレージサーバなどに保存しておく．障害発生後はバックアップから VM の復元を行う．そのため，障害が発生していない間のコストをストレージのコストのみに抑えることができる利点がある．しかし，運用系で動作させていた台数と同じ台数の VM を復旧時に同時に起動する必要があるため，復旧時間が長くなる [8]．

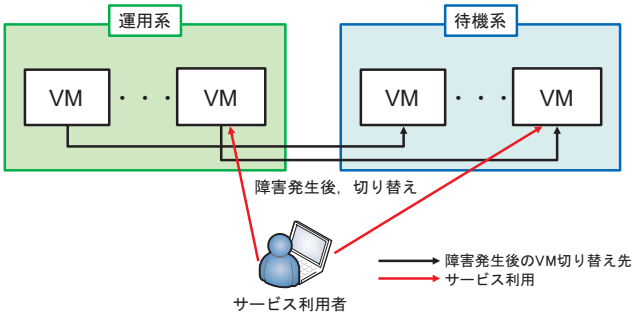

図 1 障害復旧の流れ

3. VCRecovery

本稿では，VM 内部でコンテナを用いることにより，低コストで迅速に障害復旧を行うことができるシステム VCRecovery を提案する．図 2 に VCRecovery の全体構成を示す．VCRecovery では待機系において VM 内で複数のコンテナを起動し，コンテナの中でサービスを動作させる．計算機を仮想化する VM とは異なり，コンテナは OS が提供する仮想実行環境である．クラウドでは VM に対して課金が行われるため，VM 内コンテナを用いてサービスを集約することにより，障害対策のコストを削減することができる．また，コンテナは VM と比べて軽量であるため，障害からの復旧時間を短縮することもできる．

VM 内コンテナを用いることにより，VCRecovery は低コストのウォームスタンバイを実現することができる．運用系においていくつかのサービスが複数の VM を用いて実行されている際に，待機系においては 1 台の VM の中でコンテナを用いてそれらのサービスを実行する．運用系の VM と待機系のコンテナの間では必要に応じてデータベース等の同期をとる．運用系の VM に障害が発生した際に

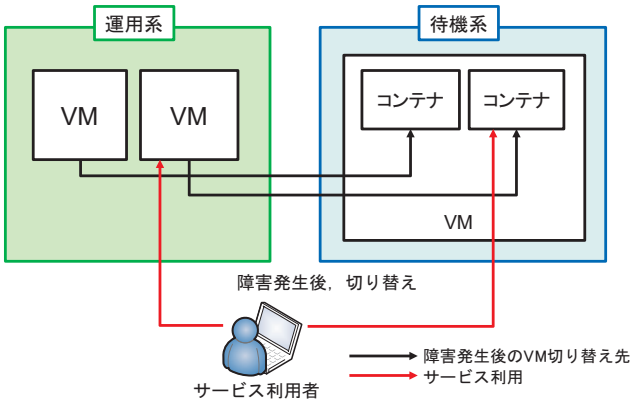

図 2 VCRecovery における障害復旧

は，待機系のコンテナに切り替えてサービスを継続する．これにより，障害対策のコストを待機系の VM 1 台分にすることができる．

VCRecovery は高速なコールドスタンバイを実現することもできる．待機系においてアイドル状態の VM を用意しておき，運用系の VM に障害が発生した際にはバックアップを用いて待機系の VM 内でコンテナを起動する．コンテナの起動時間は VM の起動時間より短いため，復旧時間を短縮することができる．待機系のアイドル状態の VM は復旧に用いる VM として様々なサービス提供者が障害対策のために共用する．そのため，待機系での共用 VM の維持費分だけ障害対策のコストは増加するが，その費用は複数のサービス提供者で折半することができる，VM を共用しない場合でも，運用系の複数の VM に対して 1 つの VM を待機系に用意すればよいため，コストの増加を抑えることが可能となる．共用 VM ではコンテナを用いてサービスを動かすため，複数のサービス提供者が利用してもセキュリティは担保される．

待機系に切り替えた後で VM の負荷が高まった場合，VCRecovery は一部のコンテナを別の VM にマイグレーションする．VCRecovery では，運用系で VM を使って動作させていた複数のサービスを，待機系ではコンテナを用いて 1 つの VM に集約する．そのため，CPU やメモリが不足したり，ディスクやネットワークの帯域が不足したりする可能性がある．これらのリソースが不足した場合には，適切なサイズの VM を起動してコンテナごとその VM にマイグレーションすることで，サービスを停止させずにリソース不足を解消することが可能である．

4. 実装

4.1 システム構成

図 3 に VCRecovery のシステム構成を示す．VCRecovery では，KVM を用いて VM を動作させ，LXD[1] を用いて VM の中でコンテナを動作させた．LXD は Ubuntu を中心に開発が行われているコンテナ型ハイパーバイザである．コンテナを提供するシステムには Docker や OpenVZ など様々なものがあるが，VCRecovery ではライブマイグレーションが可能な LXD を採用した．

障害検知・復旧には libvirt-snmp[2] と Zabbix[3] を用いた．libvirt-snmp は SNMP を用いて VM の状態を監視するためのツールである．管理システムからの要求を受信すると VM についての情報を返し，VM がクラッシュしたりすると管理システムに対して SNMP トラップを送信する．libvirt-snmp では通常の SNMP と比べて取得できる情報が限られるが，VM 内にエージェントを導入せずに VM の情報を取得できる．Zabbix は統合監視ツールであり，Zabbix サーバ，Zabbix Sender，Zabbix エージェントからなる．Zabbix を用いて，監視対象 VM の状態をグラフなどで可視化したり，状態をトリガーとしてアクションを実行したりすることができる．

4.2 障害検知

VCRecovery は VM 内のシステムの過負荷状態や異常停止などを検出するために，libvirt-snmp を用いて KVM から VM の CPU 使用量を取得する．libvirt-snmp では取得できる情報の更新間隔が 5 秒であるため，取得した CPU 使用量と 5 秒前の CPU 使用量の差分を求めることで CPU 使用率を算出する．求めた CPU 使用率は Zabbix Sender を使用して VM の UUID とともに Zabbix サーバに送信する．CPU 使用率が非常に高い状態や 0%の状態が続いていれば障害が発生している可能性がある．

VM のクラッシュを検知するために，SNMP トラップを用いた．libvirt-snmp は VM のステータスが変化すると

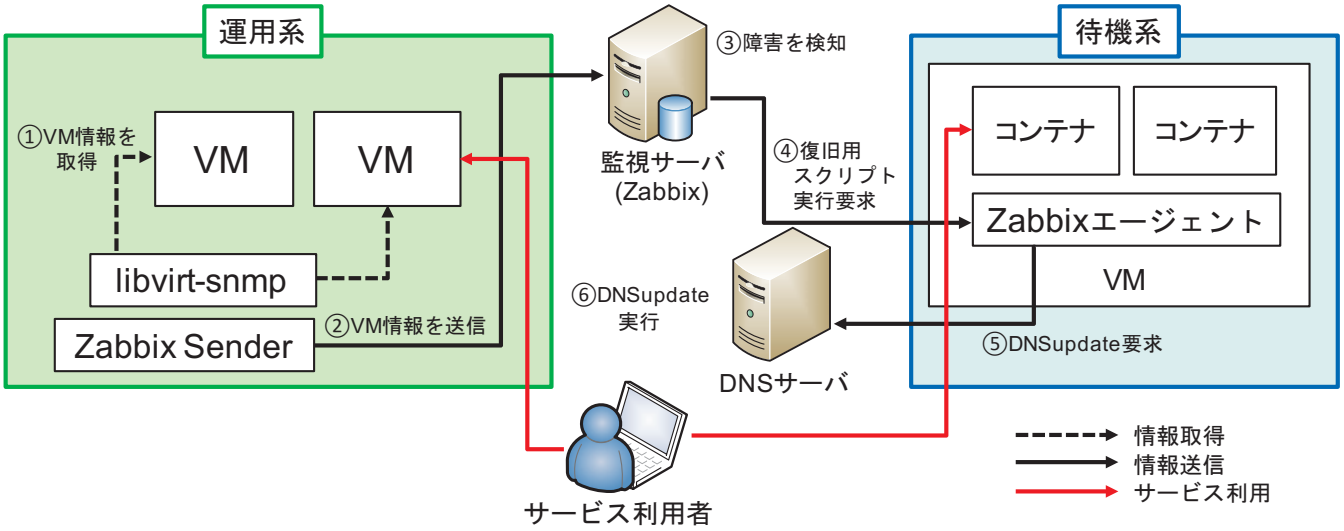

図 3 VCRecovery のシステム構成

SNMP トラップを発行する．SNMP トラップはログファイルに格納されるため，inotifywait コマンドを用いてログファイルの変更を検出する．そして，トラップから発行時刻，VM の UUID，VM の状態変化を取得し，Zabbix サーバにその情報を送信する．Zabbix サーバは VM の状態変化が shutoff の場合に VM がクラッシュしたことを検知する．

また，VM を動作させるホストやクラウドの障害を検知するために，Zabbix エージェントが取得できる情報に加え，Zabbix サーバと Zabbix エージェント間で死活監視を行う．一定時間 Zabbix エージェントからの応答がない場合，そのホストに障害が発生している可能性がある．特定のクラウド内の Zabbix エージェントからの応答がない場合には，クラウド全体に障害が発生している可能性がある．

4.3 障害復旧

VCRecovery は Zabbix と LXD，ダイナミック DNS を用いて障害復旧を行う．障害を検知すると，Zabbix サーバは復旧用 VM が動作しているホスト上の Zabbix エージェントに対してアクション実行命令を送信する．具体的には，任意のコマンドを実行させることが可能なリモートコマンド機能を利用して復旧用スクリプトを実行させる．復旧用スクリプトは必要に応じてコンテナの起動を行う．ウォームスタンバイの場合はコンテナは常に起動させておくため復旧時に起動する必要はなく，コールドスタンバイの場合にはコンテナを起動してバックアップから運用系の状態を復元する．次に，コンテナ内のサービスが使用するネットワークポートへ接続を試みる．接続が確認できると同時にサービスが再開可能と判断し，DNSupdate を実行することでサービス提供元を切り替える．DNSupdate は DNS レコードを更新し，ドメイン名に対応する IP アドレスを新しいコンテナの IP アドレスに変更する．

4.4 コンテナマイグレーション

LXD ではコンテナを停止させずに VM 間でマイグレーションさせることができる．コンテナマイグレーションはコンテナイメージを転送するストレージマイグレーションと，プロセスの状態を転送するプロセスマイグレーションからなる．ストレージマイグレーションは一旦，コンテナイメージ全体をマイグレーション先に転送し，マイグレーションの最後でもう一度同期をとる．LXD ではプロセスマイグレーションは CRIU[9] を用いて実装されている．VCRecovery の実装に用いた LXD 2.15 ではライブマイグレーションはサポートされておらず，プロセスを停止させてから状態を保存し，その状態をマイグレーション先に転送してプロセスを復元する．LXD 2.15 では WebSocket に関連した不具合のためにマイグレーションができなくなっていたため，不具合を修正してマイグレーションできるようにした．

最新の LXD 2.21 ではライブマイグレーションがサポートされている．このライブマイグレーションは CRIU の Iterative Migration を利用して実装されている．まず，プロセスを停止させずに状態を保存してマイグレーション先に転送する．2 回目以降は差分のみを保存して転送する．最後にプロセスを停止させて差分を転送し，マイグレーション先でプロセスを復元する．これにより，マイグレーション時間は長くなるが，ダウンタイムを削減することができる．ただし，現時点では LXD 2.21 を使ってライブマイグレーションを行うことができていない．

一方，CRIU では Lazy Migration と呼ばれるポストコピー・マイグレーションもサポートしている．Lazy Migration では，まず，プロセスの最小限の状態だけをマイグレーション先に転送してプロセスを復元する．プロセスがメモリにアクセスした時には Linux の userfaultfd 機構を用いてフォールトを発生させ，マイグレーション元からそのメモリページを転送する．これにより，マイグレーション時間もダウンタイムも削減することができる．我々はこれらのライブマイグレーションの利用を検討中である．

5. 実験

VCRecovery を用いて障害復旧時間を測定し，障害対策コストを見積もる実験を行った．ウォームスタンバイとコールドスタンバイを対象とし，運用系で Web サービスを提供する 4 台の VM を動作させた．その内，3 台の VM で Apache Web サーバを動作させ，1 台の VM で MySQL サーバを動作させた．待機系では 1 台の VM 内で 4 つのコンテナを動作させた．比較として，待機系でも 4 台の VM を動作させる従来システムを用いた．この実験では，運用系の VM を強制終了させてから DNSupdate の実行が完了して待機系に切り替わるまでの時間を復旧時間とした．表 1 に運用系と待機系の 2 台のマシンの構成，表 2 に VM の構成をそれぞれ示す．この実験に加え，VM 内コンテナの性能とコンテナマイグレーションにかかる時間の測定も行った．

5.1 ウォームスタンバイの復旧時間・コスト

ウォームスタンバイにおける復旧時間とコストの比較を行った．図 4 に従来システムおよび VCRecovery を用いた場合の復旧時間を示す．この実験結果から，VCRecovery において VM 内でコンテナを動作させている場合でも，VM を動作させている従来システムとほぼ同等の時間でサービスが切り替え可能であることがわかった．

図 5 に障害対策にかかるコストを示す．コストの見積もりには，Amazon EC2 で 8 個の仮想 CPU，32GB のメモリ，Red Hat Enterprise Linux の VM を 1 年間稼働させる場合の料金を用いた．この実験結果から，運用系で動作し

表 1　実験環境：運用系と待機系

項目	内容
CPU	Intel Xeon E3-1226 v3
Memory	8GB
Network	ギガビットイーサネット
OS	Ubuntu 16.04.2 LTS
Kernel	Linux 4.4.0-101-generic
Hypervisor	KVM 2.5.0
Container	LXD 2.15

表 2　実験環境：VM

項目	内容
CPU	2
Memory	2GB
OS	Ubuntu 16.04.2 LTS
Kernel	Linux 4.4.0-83-generic

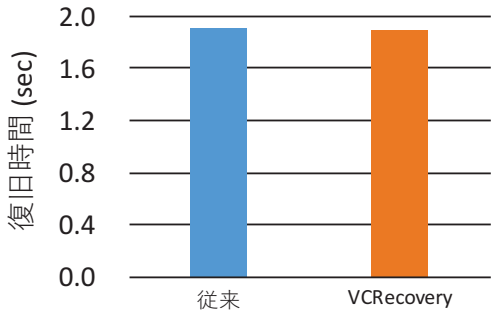

図 4　ウォームスタンバイにおける復旧時間

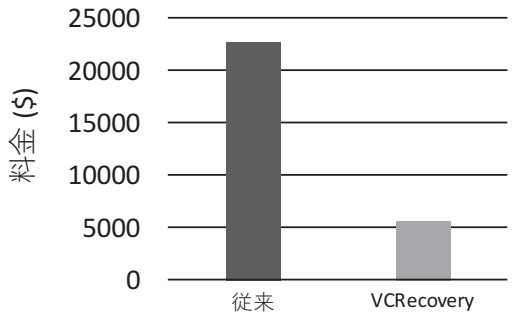

図 5　ウォームスタンバイのコスト

ている 4 台の VM と同じ台数の VM を用いて障害対策を行う場合に比べて，VCRecovery を用いて 1 台の VM のみを動作させることにより，障害対策にかかるコストは年間約 15,000 ドル削減できることが分かった．

5.2　コールドスタンバイの復旧時間・コスト

コールドスタンバイにおける復旧時間とコストの比較を行った．図 6 に従来システムおよび VCRecovery を用いた場合の復旧時間を示す．この実験結果から，VCRecovery を用いることにより復旧時間を従来システムの半分程度に

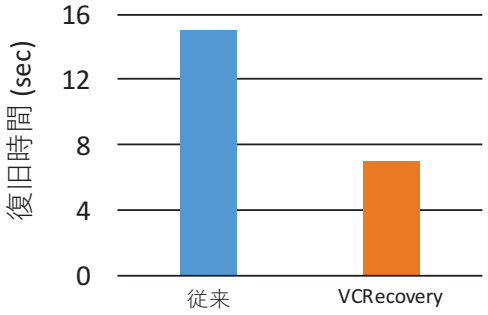

図 6　コールドスタンバイにおける復旧時間

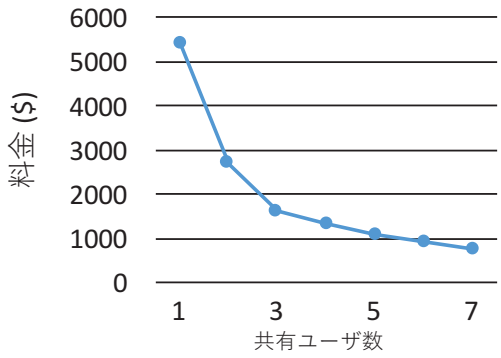

図 7　VCRecovery におけるコールドリカバリコスト

削減できることが分かった．これは，コンテナの起動時に OS を起動する必要がないため，VM と比べて短い時間で起動することができたためであると考えられる．

図 7 に VCRecovery において障害対策にかかるコストを示す．コストの見積もりには前節の Amazon EC2 の料金を用いた．従来システムでは VM のコストは 0 であったが，VCRecovery では 1 台分のコストがかかる．しかし，復旧用 VM を複数のサービス提供者で共用することができるため，VM を共用するユーザ数が増えるほど障害対策にかかるコストを削減することが可能である．共有ユーザ数が 1 から増えると急激にコストが低下するため，ある程度のユーザ数で共有すれば十分にコストを抑えられることが分かった．

5.3　VM 内コンテナの性能

ベンチマークを用いて VM とその中で動作する LXD コンテナの性能を測定し，VM 内でコンテナを用いることによる性能低下を調べた．LXD は様々なストレージバックエンドをサポートしているため，ディレクトリ（dir）と ZFS を用いて実験を行った．

図 8 に UnixBench の結果を示す．多くの項目で VM と VM 内コンテナは同等の性能であったが，いくつかの項目で性能の違いが顕著に表れた．VM 内コンテナのコンテキストスイッチの性能は VM の約 2 倍となったが，システム

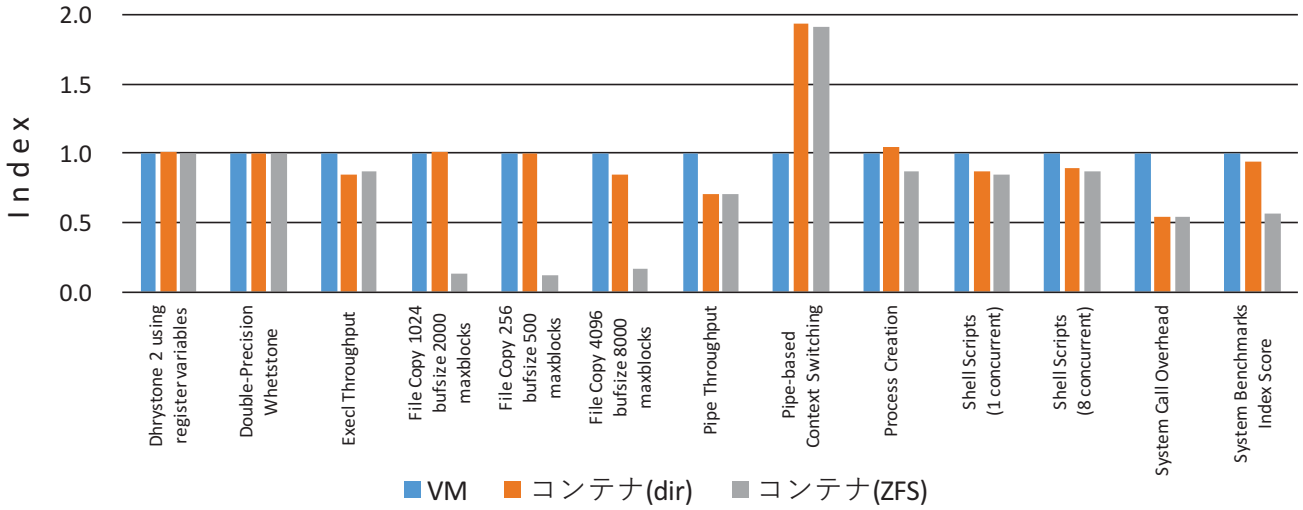

図 8 VM 内コンテナの性能（UnixBench）

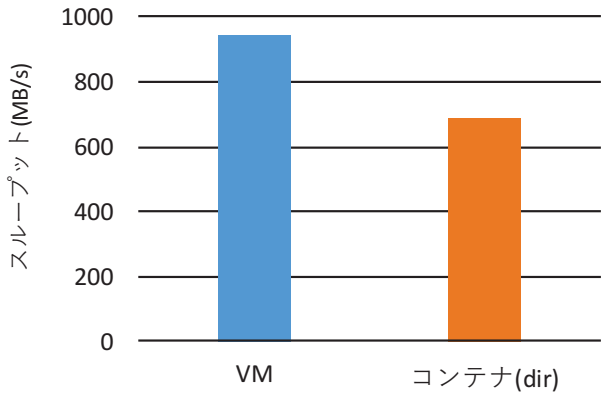

図 9 VM 内コンテナの性能（fio）

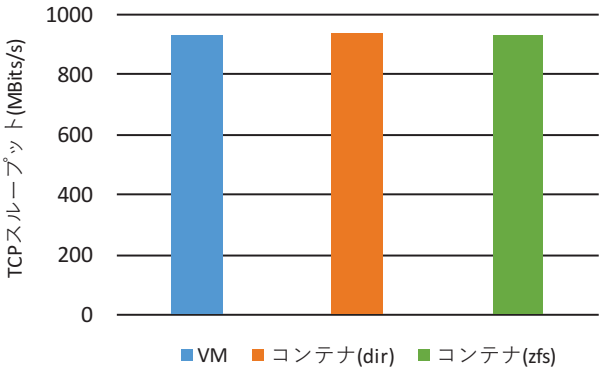

図 10 VM 内コンテナの性能（iperf3）

コールの性能は VM の約半分となった．ストレージバックエンドの影響として，ファイルコピーでは ZFS を用いた場合の性能だけが大幅に低下した．トータルスコアを比較すると，ストレージバックエンドにディレクトリを用いた場合，VM 内コンテナの性能低下は 6.6%であった．一方，ZFS を用いた場合の性能低下は 43.1%になった．

次に，fio を用いてディスクアクセスのスループットを測定した．図 9 に結果を示す．ストレージバックエンドにディレクトリを用いた場合の性能低下は 28%であった．一方，ZFS を用いた場合はエラーが発生して測定を行うことができなかった．また，iperf3 を用いて TCP スループットも測定した．図 10 に結果を示す．結果から，VM 内コンテナを用いることによる TCP スループットの低下はなかった．

5.4 VM 内コンテナのマイグレーションの性能

VM 内コンテナをマイグレーションする際の性能を調べた．この実験では，2 台のホスト上で動作する VM 間でコンテナをマイグレーションし，マイグレーション時間とダウンタイムを測定した．比較として，VM を用いずにホスト上で直接コンテナを動作させてマイグレーションを行った場合にもマイグレーション時間を測定した．この実験に用いたコンテナのイメージサイズは 950MB であった．また，他の実験とは違い，最新の LXD 2.21 を用いた．

図 11 にマイグレーション時間を示す．VM 内でコンテナを動作させた場合には 1 秒程度，高速化していることが分かる．この原因は不明であるが，VM 内コンテナのオーバヘッドはないことが分かった．一方，ライブマイグレーションを行えなかったため，VM 内コンテナのダウンタイムは 5.6 秒とかなり長くなった．これはライブマイグレーションを行うことができれば改善されると考えられる．

6. 関連研究

サービスを少数の VM に集約するシステムはいくつか提案されている．Picocenter[10] は，長期間実行されるがほとんど使われないサービスを VM 内コンテナを用いて実行する．サービスが使用されず，コンテナが非アクティブになった場合は，コンテナごとストレージにスワップアウ

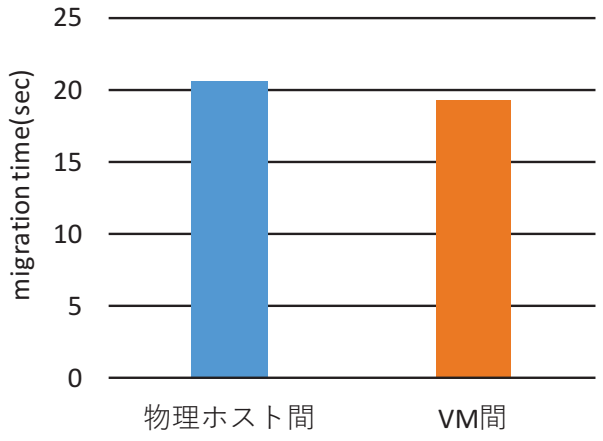

図 11 コンテナのマイグレーション時間

トし，サービスへの要求が来た時にコンテナをスワップインする．障害が発生するまで利用されないサービスを動かすのに VM 内コンテナを利用するという点で VCRecovery は Picocenter に似ている．コールドスタンバイを用いる際にコンテナをスワップアウトしておき，障害復旧時に高速にスワップインする Picocenter の機構が利用可能である．

FlexCapsule[11] では，VM 内で軽量な VM を用いてサービスを実行する．クラウドにおいてきめ細かい VM 構成の最適化を動的に行うことにより，コストを削減することが可能である．FlexCapsule では，ネストした仮想化を用いて VM の中で VM を動作させ，ライブラリ OS を用いることで軽量化している．しかし，VM 内で VM を動作させるオーバヘッドがまだ大きい．

ホットスタンバイのために VM 間で同期を取る技術がいくつか提案されている．Remus[12] は，運用系で動いている VM の CPU やメモリの状態の差分を待機系に送り，障害が発生すると最新の状態に保たれている待機系の VM に切り替える。ネットワーク送信やディスク書き込みは同期が完了するまでバッファリングされる．Kemari[13] は，VM からネットワーク送信やディスク書き込みを行う際にだけ VM の同期をとることにより，同期の頻度を減らしている．COLO[14] は，運用系で受信した要求パケットを運用系と待機系の両方の VM に配送し，これらの VM からの応答パケットが一致するまで待機させることで同期をとる．VCRecovery ではコンテナと VM の間で同期をとる必要がある．

Swift Birth/Quick Death[8] では，Xen における複数の VM の同時起動を高速化している．コールドスタンバイにおいて障害復旧の際には多数の VM が同時に起動されるため，この機構により復旧時間を短縮することができる．しかし，VM の起動時間は短縮することができるが，それ以降の OS の起動時間は短縮することができない．また，我々の実験によると，KVM の場合には複数の VM を同時に起動しても Xen のようなボトルネックは見つからなかった．

文献 [15] では，クラウドを用いた災害復旧について議論し，災害発生後にサービス提供を再開するために必要なコストを分析している．その結果，アプリケーションによってはクラウドを待機系として用いることで大きくコストを抑えられることが示されている．

7. まとめ

本稿では，VM 内部でコンテナを用いることにより低コストで迅速に障害復旧を行うことができるシステム VCRecovery を提案した．VCRecovery では，待機系の 1 つの VM の中に軽量なコンテナを集約するため，障害が発生するまでの間はウォームスタンバイのコストを抑えることができる．また，コンテナは VM よりも短時間で起動することができるため，コールドスタンバイでも迅速に復旧を行うことができる．復旧後に VM の負荷が高まった場合はコンテナをより大きな VM へとマイグレーションすることで性能を改善する．LXD や Zabbix などを用いて VCRecovery を実装し，障害検知から復旧およびコンテナのスケールアップまでを自動で行うことができることを確認した．また，実験により，VCRecovery では従来システムに比べて復旧時間を短縮することができ，コストを抑えられることが分かった．

今後の課題は，運用系の VM と待機系の VM 内コンテナとの間でディスクなどの VM の状態の同期をとることが挙げられる．運用系でも VM 内コンテナを用いてコンテナ間での同期を取るか，VM とコンテナの間で同期を取れるようにするかを検討中である．また，コンテナマイグレーションを実用的に使えるようにするために，性能を改善する必要がある．本稿では，VM 単位で障害が発生した場合における障害対策を主に考えたが，クラウドでは様々な規模で障害が発生するため，障害規模に応じた障害対策を実現することも今後の課題である．

参考文献

[1] Linux Containers, https://linuxcontainers.org（Jul. 11, 2017 参照）
[2] Libvirt-snmp, https://wiki.libvirt.org/page/Libvirt-snmp（Sep. 26, 2017 参照）
[3] Zabbix Official, https://www.zabbix.com（Jan. 23, 2018 参照）
[4] B. Wellington, “ Secure Domain Name System（DNS）Dynamic Update, ” IETF, RFC 3007, Nov. 2000.
[5] Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region, https://aws.amazon.com/message/41926/（Nov. 30, 2017 参照）
[6] Summary of the AWS Service Event in the Sydney Region, https://aws.amazon.com/jp/message/4372T8/（Nov. 20, 2017 参照）

[7] AWS Disaster Recovery, https://aws.amazon.com/disaster-recovery/（May 23, 2017 参照）
[8] V. Nitu, P. Olivier, A. Tchana, D. Chiba, A. Barbalace, D. Hagimont, and B. Ravindran: Swift Birth and Quick Death: Enabling Fast Parallel Guest Boot and Destruction in the Xen Hypervisor, In Proc. ACM SIGPLAN/SIGOPS Int. Conf. on Virtual Execution Environments, pp.1-14, 2017.
[9] CRIU, https://criu.org/Main_Page（Jan. 14, 2018 参照）
[10] L. Zhang, J. Litton, F. Cangialosi, T. Benson, D. Levin, and A. Mislove: Picocenter: supporting long-lived, mostly-idle applications in cloud environments, In Proc. European Conference on Computer Systems, pp.37. 2016.
[11] K. Kourai, K. Sannomiya: Seamless and Secure Application Consolidation for Optimizing Instance Deployment in Clouds, In Proc. Int. Conf. Cloud Computing Technology and Science, pp.318-325, 2016.
[12] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield: Remus: high availability via asynchronous virtual machine replication, In Proc. 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI'08), 2008.
[13] 田村 芳明, 柳澤 佳里, 佐藤 孝治, 盛合 敏: Kemari：仮想マシン間の同期による耐故障クラスタリング, 情報処理学会論文誌, ACS, Vol.3, No.1, pp.13-24, 2010.
[14] Y. Dong, W. Ye, Y. Jiang, I. Pratt, S. Ma, J. Li, and H. Guan: COLO: COarse-grained LOck-stepping virtual machines for non-stop service, In Proc. 4th annual Symposium on Cloud Computing (SOCC'13), 2013.
[15] T. Wood, E. Cecchet, K. K. Ramakrishnan, P. Shenoy, J. Van der Merwe, and A. Venkataramani: Disaster recovery as a cloud service: economic benefits & deployment challenges, In Proc. 2nd USENIX Conference on Hot topics in cloud computing (HotCloud'10), 2010.