Vol.2015-OS-134 No.2
2015/8/4

冗長構成された仮想マシンの同期処理における負荷低減方式の提案

笠江 優美子[1,a)]　車谷 駿介[1,b)]　森下 慎次[1,c)]　高橋 寛幸[1,d)]　日高 東潮[1,e)]

概要：仮想マシンモニタ (VMM) で実現する耐故障技術では，異なる物理マシン上の 2 つ以上の仮想マシン (VM) 間で冗長構成を維持するための処理が必要である．しかし，この処理がオーバヘッドとなり，既存の冗長構成維持の方式では，耐故障技術を適用しない場合と比較し VM 上のアプリケーションが行う処理のスループットが低下してしまうという問題がある．このことから，耐故障技術のための低オーバヘッドな冗長構成維持の手法を実現することが重要である．これまで我々は，既存の VMM で実現する冗長構成維持の処理方式の中で，高効率であると報告されている COarse-grained LOck-stepping (COLO) について，KVM をベースに再現し特性評価を行っている．その結果，COLO は VM 上の冗長構成維持の対象となるアプリケーションがシングルプロセスで処理される場合において低オーバヘッドで冗長構成を維持することができるが，対象となるアプリケーションがマルチプロセスで処理される場合は，オーバヘッドが大きくなる可能性があることを確認している．本論文では，COLO をベースとし，マルチプロセス処理時に増加する非決定性への対処を追加した冗長構成を維持する方式である Buffered-Compare を提案する．本評価環境において，128 プロセスでの並列処理を行ったところ，本提案方式では既存方式の 1.3 倍のスループットが得られ，冗長構成を維持することによるオーバヘッドが削減されることを確認した．

1. はじめに

近年，仮想化技術の進展に伴い，仮想マシン (VM) がサービス基盤として普及が進んできており，DB サーバのような高い無停止性が要求されるようなアプリケーションにおいても，VM 上で構築・運用されるようになりつつある．そのようなアプリケーションに対し無停止性を実現する技術としては，アプリケーションで冗長構成を組み可用性を高めるものがあるが，VM 上で無停止が要求されるアプリケーションは多岐にわたるため，アプリケーションとは独立して無停止性を実現するための仕組みが重要である．

そのような仕組みを実現する技術の 1 つとして，特別なハードウェアを利用せず，ソフトウェアでハードウェア故障によるアプリケーションの停止を回避することができる，ソフトウェアでの耐故障技術がある．本研究では，VM 上で動く OS やサービスに手を加えずに無停止性を実現することができるという観点から，仮想マシンモニタ (Virtual Machine Monitor:VMM) での耐故障技術に注目する．

[1] 日本電信電話株式会社 NTT ソフトウェアイノベーションセンタ
[a)] kasae.yumiko@lab.ntt.co.jp
[b)] kurumatani.shunsuke@lab.ntt.co.jp
[c)] morishita.shinji@lab.ntt.co.jp
[d)] takahashi.hiroyuki@lab.ntt.co.jp
[e)] hitaka.toshio@lab.ntt.co.jp

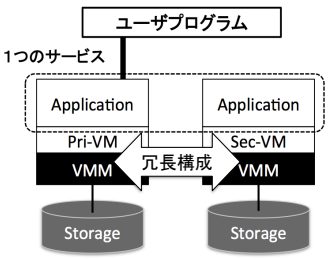

図 1 VMM での耐故障技術 (通常時)

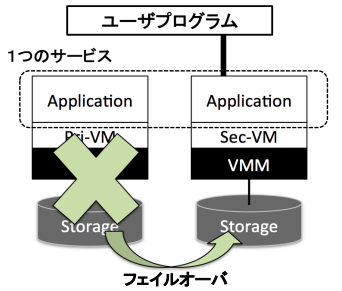

図 2 VMM での耐故障技術 (故障発生時)

VMM で実現する耐故障技術では，図 1 に示すように，異なる物理マシン上の 2 つ以上の VM 間で冗長構成を維持する．ユーザプログラムと通信をしている VM(Primary VM : Pri-VM) が動作する物理マシンで故障が発生した場合，別途用意される故障検出の仕組みにより Pri-VM の外部通信が切断され，図 2 に示すように瞬時に異なる物理マシンで動く VM(Secondary VM : Sec-VM) へユーザプログラムからのネットワーク接続セッションを含めフェイルオーバを行うことで，ユーザプログラム側には故障発生を隠ぺいする．

このように VMM では，故障発生に備え，故障が検出されていない時において Pri-VM と Sec-VM を常に同一状態 (本論文では，同一状態を “ Pri-VM から Sec-VM へフェイ

ルオーバが行われることに対して，外部にあるユーザプログラム側に特別に対応させる必要がない状態 ” と定義) に保つという冗長構成を維持する処理が必要である．既存の VMM で実現する耐故障技術 [1][2][3][4] の多くは，冗長構成を維持する処理に際し VM のライブマイグレーションの仕組みを利用して実現している．しかし，この冗長構成維持の処理自体がオーバヘッドとなり，耐故障技術を適用しない場合と比較して VM 上の処理のスループットが低下してしまうという問題がある [2]．そのため，耐故障技術のための低オーバヘッドな冗長構成維持を実現することが重要である．

本研究では，既存の VMM レイヤで実現する冗長構成維持の方式の中で高効率であると報告されている，COarse-grained LOck-stepping(COLO)[5] という方式に注目した．これまで我々は，COLO のアルゴリズムの評価を実施している [6]．評価の結果，VM 上のアプリケーションがシングルプロセスで処理される場合においては，COLO によって低オーバヘッドで冗長構成を維持することができるが，近年利用される多くのアプリケーションのようにマルチプロセスで処理がなされた場合は，VM 上の OS のスケジュールの非決定性等が影響し，COLO においても冗長構成維持のオーバヘッドが大きくなってしまう場合があるという課題を示した．

本論文では，上記課題を解決するための一方式である Buffered-Compare を提案する．Buffered-Compare は，COLO をベースにマルチプロセス処理時に発生する OS の非決定性等の影響を考慮するアルゴリズムを追加している．本論文では，COLO と Buffered-Compare の評価を行い，Buffered-Compare が COLO と比較してマルチプロセス処理時においても低オーバヘッドで冗長構成が維持できるということを示す．

本論文の前提として，故障検出手段が検出する故障は，heartbeat 等の既存の故障検出手段で瞬時に検出可能である電源断等の単一のハードウェア故障とし，故障検出手段の実現については対象外とする．また，近年のアプリケーションは，クライアントサーバモデルのような，外部から何らかの入力を受け取って処理を行い，外部にその処理結果を出力するものが多いため，本論文ではこのような利用形態のアプリケーションをターゲットとしている．

2. COarse-grained LOck-stepping (COLO)

2.1 COLO の処理フロー

COLO の処理フローについて，図 3 を用いて説明する．

COLO では，各 VM 起動時において初期同期を実行し，メモリ，CPU レジスタ等の内部状態や MAC アドレス等を 2 つの VM 間で同一状態とした上で，冗長構成維持の処理を開始する．まず Pri-VM がユーザプログラムから入力パケットを受け取ると，COLO によってパケット複製される (図 3 中①)．そして複製したパケットを Sec-VM へも転送し，各 VM は受け取った入力パケットの元，処理を行う (②)．COLO では基本的に，同一状態の VM へ同じ入力パケットを与えることで，各 VM で同じ処理がされることを期待する．しかし，同一状態の VM へ同一の入力パケットを与えたとしても，入力による処理や VM 内の内部処理によって非決定的な処理が行われた場合，VM 間で状態の違いが生じる可能性がある．例えば，処理に乱数発生や時刻情報に関する問い合わせが含まれていた場合，同じ処理がなされない可能性が考えられる．そこで，COLO にはその VM 間の状態の違いをネットワーク (NW) 出力の図 3 中④の契機で検出する仕組みがある．COLO では，Pri-VM と Sec-VM のユーザプログラムへの出力パケットを Primary の VMM 内の Compare Module という箇所に集め (③)，それぞれのパケット内容を，到着した順に順次比較していく (④)．パケットの比較の結果，その内容が一致していた場合ユーザプログラム側へ返答される．一致していなかった場合，ディスクの同期やチェックポイント (CP) 転送処理という VM のライブマイグレーションの仕組みを利用したメモリやレジスタ等のデータ転送処理を実施することで 2 つの VM の内部状態を強制的に同一にする (⑤⑥)．その後，ユーザプログラム側へ Pri-VM が返答する．

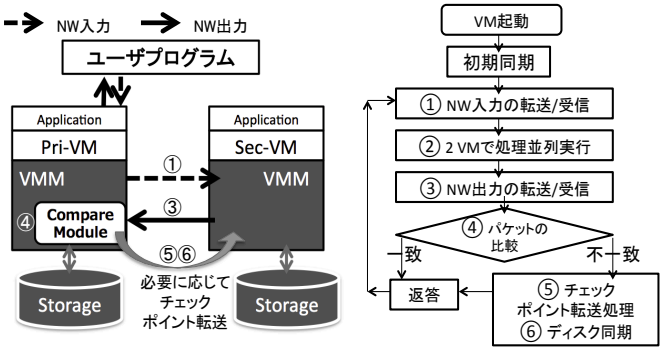

図 3 COLO の処理フロー

このように COLO では，ユーザプログラムに返答する契機である NW 出力のたびに Compare Module で返答内容が同一であることを確認をすることで，2 つの VM で同一の処理が行われたことを保証する．また，一定間隔以上 (デフォルト値で 10 秒)CP 転送処理が行われなかった場合，強制的に CP 転送処理を実施する．このようにすることで，2 つの VM 間が極端に異なる状態となることを防止し，転送するメモリ差分の量に依存する CP 転送処理の負荷を一定以内に収めている．

2.2 COLO による冗長構成維持と他方式との比較

Remus[1] や Kemari[2][3][4] などの VMM での冗長構成

維持の方式の多くは，任意の契機で CP 転送処理を行うことで，Pri-VM と Sec-VM の内部状態を同一とする方式であった．つまり，“2 つの VM のメモリやレジスタ等の内部状態の同一性 ” を保証することでフェイルオーバに備えていた．しかし，CP 転送処理は，Pri-VM を一度停止させ，データ転送を行い，各 VM を再開するという処理を含む．また，常に内部状態の同一性を確保するのためには，少なくとも Pri-VM の内部状態が変化するイベントが発生するたびに CP 転送処理を実施する必要がある．そのため，頻繁な CP 転送処理を実施する場合があり，CP 転送処理自体がオーバヘッドの要因となっていた．

これに対し COLO では，内部状態の同一性を常に保証するのではなく，“2 つの VM の出力の同一性 ” を保証している．VM の出力結果のみを扱うユーザプログラムの視点で見たときには，この同一性においても十分な冗長構成が維持できる．なぜならば，ユーザプログラムにとって VM 内部の状態は単なるブラックボックスにすぎないからである．1 章で述べたような本研究で対象とする形態のアプリケーションでは，NW 出力へ影響しない限り，2 つの VM 間でどのメモリ番地にどの情報を保持しているのかは同一でなくても構わない．ただし，ユーザプログラムが，特定の情報を保持するメモリ番地を問い合わせ，2 つの VM からの返答に不一致が生じた場合，CP 転送処理により内部状態を同一にすることで状態の違いを解消する仕組みである．

このように COLO は，冗長構成に関する VM 間の同一性の厳密さを緩和し，CP 転送処理自体の実施頻度を削減することで，KVM・Xen ベースの冗長構成を維持する方式の中でも高効率な方式であると考えられている [5][6].

2.3 COLO による冗長構成維持の問題点

我々の COLO の評価において，VM 上のアプリケーションがシングルプロセスで処理される場合，COLO は高効率で冗長構成維持が可能であることを確認した [6]．しかし，近年利用される多くのアプリケーションのようにマルチプロセスで処理がなされた場合は，COLO においても冗長構成維持のオーバヘッドが大きくなってしまう場合があるということもわかっている [6].

我々は，上記の場合オーバヘッドが大きくなってしまう要因の 1 つとして，VM 上の OS のスケジューリングの非決定性があると考えている．各 VM へ NW 入力が到着すると，OS はそれらを処理するためのスケジューリングを行う．そのスケジューリングに際し，各 VM で全く同一のスケジューリングが行われるとは限らない．通常のスケジューラでは，コンテキストスイッチを実行させ多くのプロセスを並列処理していくが，タイマーなどの割り込み等が原因となり，処理順序が各 VM で異なる可能性がある．これにともなって，NW 出力順序も同一にならないと考えられる．この OS のスケジューリングの非決定性は，より多くのプロセスを同時走行させる際に影響が拡大する．

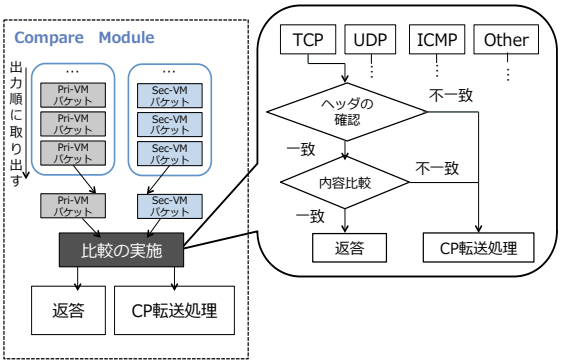

図 4 COLO における比較アルゴリズム

COLO の Compare Module での比較アルゴリズムは，図 4 に示すように，各 VM からの NW 出力パケットを管理するキューから出力順にそれぞれ取り出され，パケットのヘッダ情報の確認や処理結果内容を順次比較するというナイーブなアルゴリズムである．そのため，ただ単に各 VM からの出力順序のみが異なる場合においても，パケット内容の不一致と判定され，本来ならば必要のない CP 転送処理が実施されている可能性がある．事実，我々の評価実験 [6] では，マルチプロセス処理時において，各 VM からの出力パケット内容自体の不一致に比べ，各 VM からの NW 出力順序の入れ替りの方が多く確認されている．このことが，マルチプロセス処理時に出力パケット内容の不一致判定による不必要な CP 転送を頻発させ，オーバヘッドの要因となっていたと考えることができる．

3. 提案方式: Buffered-Compare

本論文では，COLO の問題点を解決する一方式として，VM 上の OS のスケジューリング等の非決定性への対処を追加した比較アルゴリズムである Buffered-Compare を提案する．本方式によって，VM 上のアプリケーションが近年の多くのアプリケーションのようにマルチプロセスで処理を行った場合においても，CP 転送処理の実施頻度を抑え，低オーバヘッドでの冗長構成維持の実現を目指す．

3.1 Buffered-Compare とは

COLO の比較アルゴリズムでは，前述の通り各 VM からの NW 出力順序が同一ではないことにより CP 転送処理が頻発したことが，マルチプロセス処理時においてオーバヘッドが大きくなる要因であると考えられる．そこで Buffered-Compare では，各 VM からの NW 出力順序の入れ替わりを考慮する．つまり，Buffered-Compare では，Pri-VM と Sec-VM の NW 出力内容の不一致を検出した場合，即座に CP 転送処理を実施するのではなく，Sec-VM の出力パケットを管理しているキューから，不一致判定され

た出力パケットの次の出力パケットを取り出し，再度比較を実施する．これを繰り返すことによって，NW 出力順序の入れ替わりによる不一致判定を回避し，不要な CP 転送処理の実施によるオーバヘッドを削減することができる．

しかし，Buffered-Compare によって再比較を繰り返し行っている間は，当該出力パケットはユーザプログラムへ返答されないこととなってしまう．例えば，処理の違いによって真に出力パケット内容が不一致になっていた場合においても，Buffered-Compare の比較アルゴリズムによる再比較を繰り返してしまい，いつまでもユーザプログラムへ返答されないことになる．そのため，Buffered-Compare では，再比較を繰り返す閾値を設定する．閾値は時間で規定され，Pri-VM の出力パケットを管理しているキューから出力パケットを取り出した瞬間からの経過時間で判定する．経過時間が閾値以下である場合は，Buffered-Compare による再比較を繰り返し，閾値より大きくなると CP 転送処理を実施する．これによりユーザプログラムに対する返答を保証することができる．

また，Buffered-Compare においても，COLO と同様に一定間隔以上 (デフォルト値で 10 秒)CP 転送処理が行われなかった場合，強制的に CP 転送処理を実施する．これによって 2 つの VM 間が極端に異なる状態となることを防止し，転送するメモリ差分の量に依存する CP 転送処理の負荷を一定以内に収めている．

3.2 Buffered-Compare のアルゴリズム

図 5 に，Buffered-Compare のアルゴリズムを示す．

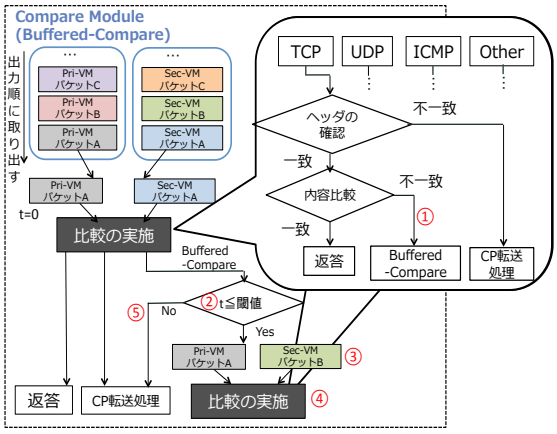

図 5 Buffered-Compare のアルゴリズム

(1) 各 VM からの出力パケットを管理しているキューから先頭を取り出す．この時，Pri-VM の出力パケットをキューから取り出した時からの経過時間 (図 5 中では t) を測定するため，この値を初期化する．
(2) 各 VM からの出力パケット内容の比較を行う．
(3) 出力パケット内容が一致していれば (10) へ．(COLO と同様)
(4) 出力パケット内容の不一致と判定された場合には，NW 出力順序の入れ替りである可能性がある．Buffered-Compare による比較アルゴリズムに移行する (図 5 中①)．
(5) Pri-VM の出力パケットをキューから取り出してからの経過時間 t が閾値以下であるか判定する (図 5 中②)．
(6) 経過時間 t が閾値より大きい場合には (9) へ (図 5 中⑤)．
(7) 経過時間 t が閾値以下の場合には，不一致と判定された Sec-VM の出力パケットの，次の出力パケットを Sec-VM のキューから取り出す (図 5 中③)．この時，不一致と判定された Sec-VM の出力パケットは，キュー内の元の場所に戻される．
(8) Pri-VM の出力パケットと，新たに取り出した Sec-VM の出力パケットとで再度比較 (図 5 中④) すべく (2) へ．
(9) 保持している出力パケットをユーザプログラムへ返答し，CP 転送処理を実施する．Pri-VM と Sec-VM を同一状態にした後に (1) へ．
(10) 当該パケットをユーザプログラムへ返答し (1) へ．

4. 実装

本研究では，VMM として更なる普及が期待される KVM[9] での低オーバヘッドな冗長構成維持の手法の実現を目指している．COLO のソースコードは，KVM での利用を想定したものが OSS として公開されている [11]．COLO は，主にエミュレーションレイヤである QEMU[10] に対し機能追加されているが，Compare Module については Pri-VM が動く物理マシン上の OS のカーネルモジュールとして実装されている．本研究では，COLO v1.2-basic[12] のブランチをベースとし Buffered-Compare を Compare Module の一機能として実装している．

5. 評価実験

5.1 概要

Buffered-Compare の効果を示すため，COLO との比較評価を行った．本実験では，VM 上で PostgreSQL 9.4[13] のデータベースサーバを立ち上げ，COLO で冗長構成を維持した場合と，Buffered-Compare を適用した場合の比較評価を行った．PostgreSQL を含む一般の DBMS は耐故障性が重要であるため，本評価における VM 上のアプリケーションとして適切であると考える．本実験では，PostgreSQL 上の擬似アプリケーションとして pgbench[14] を利用した．pgbench は PostgreSQL の標準的なベンチマークであり，TPC-B[15] を模している．pgbench では，データベースサーバへ同時接続するクライアント数が任意に指定できる．本実験では，同時接続クライアント数を変化させ評価を行った．PostgreSQL は 1 クライアントは 1 プロセスとして処理されるため，同時接続クライアント数は VM 内で

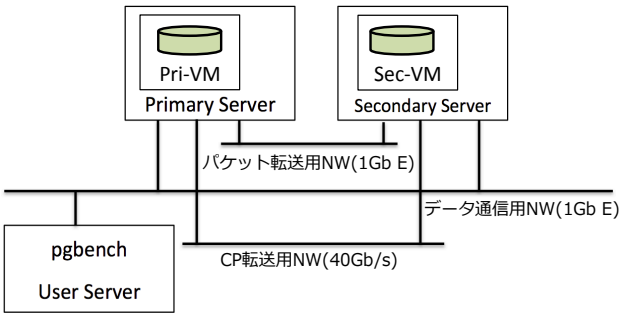

図 6 実験環境

表 1 Primary Server

サーバ	Quanta S200-X12RS
OS	Ubuntu 14.04 (3.18.11)
CPU	Intel(R) Xeon(R) 4 core CPU E5-2306v2 @1.8GHz
メモリ	Samsung M393B2G70DB0-YK0 16Gbyte × 2
HDD	SEAGATE ST1200MM0017
NIC	Intel I350 Gigabit Ethernet × 2
InfiniBand	Mellanox MT26428

表 2 Secondary Server

サーバ	Tyan B7066G24W4H
OS	Ubuntu 14.04 (3.18.11)
CPU	Intel(R) Xeon(R) 4 core CPU E5-2306v2 @1.8GHz
メモリ	Samsung M393B1G70QH0-CMA 8Gbyte × 4
HDD	Western Digital WD9001BKHG-02D22
NIC	Intel I350 Gigabit Ethernet × 2
InfiniBand	Mellanox MT26428

表 3 仮想マシン

VMM	KVM(QEMU 2.3.50)
OS	Ubuntu 14.04 (3.13.0-24-generic)
vCPU	4
メモリ	4 Gbyte
ディスク	64 Gbyte

の PostgreSQL への同時走行プロセス数と同等となる．

5.2 評価方針

Buffered-Compare では，各 VM からの NW 出力順序の入れ替りを考慮することができるため，CP 転送処理の実施頻度を削減し，COLO と比較して低オーバヘッドで冗長構成維持ができることを期待する．しかし，COLO と比較して，Buffered-Compare のアルゴリズムによるレイテンシの増加や，CP 転送処理の実施頻度の削減による 1 回の CP 転送処理にかかるデータ転送量等の増加が考えられる．

本評価実験では，COLO と比較した Buffered-Compare の効果及び影響を評価した．

5.3 実験環境

図 6 に，本評価の実験環境を示す．本実験環境では，表 1 及び表 2 に示す物理マシン 2 台で評価を行った．図 6 に示すように，物理マシン間は 2 本のインターコネクトで接続されており，1 本はユーザプログラムから Pri-VM への NW 入力を Sec-VM にも転送する 1GbE のインターコネクト，もう 1 本は CP 転送処理に用いられる 40Gb/s の IP over InfiniBand のインターコネクトである．また，各物理マシン上では表 3 に示す VM を起動させる．VM 上の PostageSQL のデータベースはスケールファクタ 5000 で生成し，50Gbyte ほどの大きさである．また，定期的な CP 転送間隔はデフォルト値である 10 秒とした．

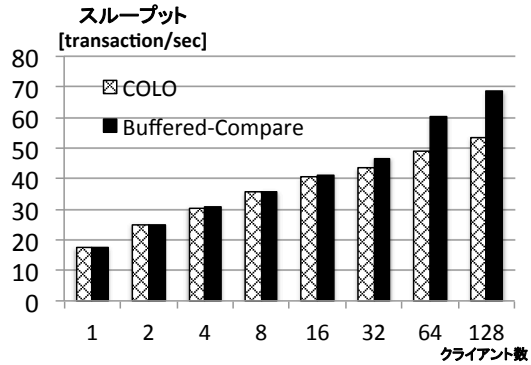

図 7 スループット測定結果

Buffered-Compare では，再比較を継続する期間を指定する閾値を設定する必要がある．本実験では，1.5 マイクロ秒とした．この値は，本実験環境における各 VM からの NW 出力の順序の入れ替り具合 (3，4 パケット間での入れ替わりが多発）から，ヒューリスティックに決めた数値である．この閾値の最適化に向けた検討については今後の課題とする．

5.4 評価実験結果:スループット

図 7 に，本評価実験におけるスループットの測定結果を示す．横軸が同時接続クライアント数，縦軸がスループットである．また，図 8 に，各同時クライアント接続数ごとの CP 転送処理の実施回数を示す．図 8 では，Compare Module により不一致判定されたことによって CP 転送処理が実施された回数のみを示しており，定期的な CP 転送処理の実施回数についてはカウントしていない．

図 7 から，同時接続クライアント数の増加に関わらず，Buffered-Compare が COLO の比較アルゴリズムに比べ同等もしくは高いスループットであることがわかる．また，図 8 より，COLO では同時接続クライアント数の増加にともなって Compare Module によって不一致判定されたことによる CP 転送処理の実施回数も増加していることがわかる．これは前述のとおり，マルチプロセス処理による OS の非決定性によって各 VM からの NW 出力順序の入れ替りが発生し，不一致判定が頻発しているためだと考える．特に同時接続クライアント数が 32 以上の場合においてこれが顕著である．この不一致発生について，Buffered-Compare

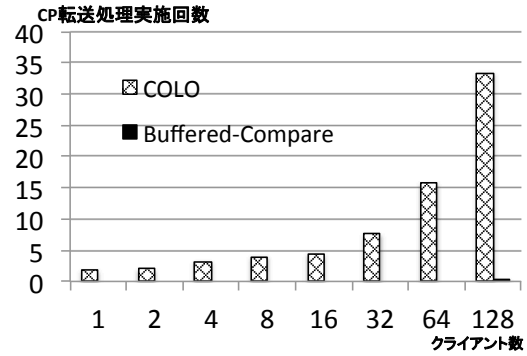

図 8 Compare Module での不一致判定による CP 転送処理実施回数

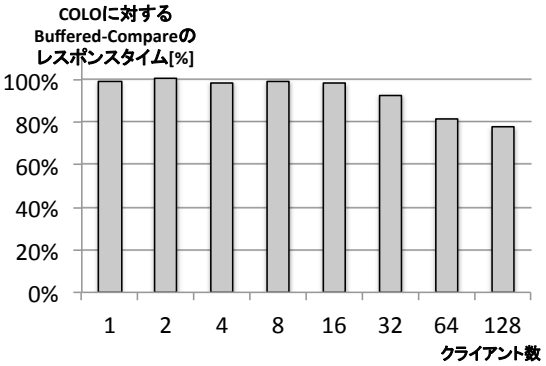

図 9 平均レスポンスタイム

では各 VM からの NW 出力の入れ替りを考慮する効果により，Compare Module での不一致判定による CP 転送処理の実施頻度を削減し，ほぼ定期的な CP 転送処理のみとなったと考えられる．本評価実験では，Buffered-Compare により COLO の最大 1.3 倍のスループットが得られ，冗長構成を維持することによるオーバヘッドが削減されることを確認した．

5.5 評価実験結果:レスポンスタイム

Buffered-Compare では，アルゴリズムの特性上，COLO に比べてレイテンシが増加することが考えられる．これは，Buffered-Compare で再比較を繰り返し行った場合，COLO の比較アルゴリズムに比べて遅延が発生するためである．本実験では，レイテンシの増加がレスポンスタイムに与える影響を考察するため，各同時接続クライアント数において，1 クライアントが全トランザクションを終えるまでの平均レスポンスタイムの評価を行った．

図 9 に，Buffered-Compare と COLO のレスポンスタイムの比較を示す．グラフでは，各同時接続クライアント数ごとの COLO のレスポンスタイムを基準とした，Buffered-Compare のレスポンスタイムの割合を示している．

このグラフから，同時接続クライアント数が 32 より小さい場合においては，レイテンシの増加がレスポンスタイムに与える影響は少ないことが確認できた．また特に

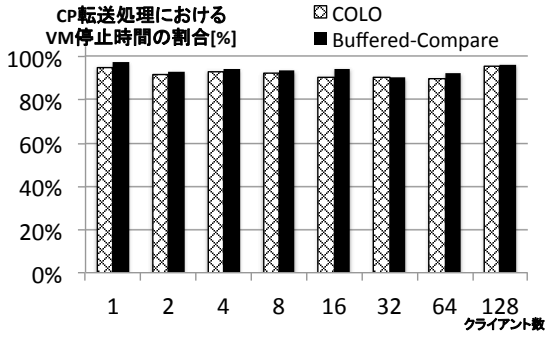

図 10 1 回の CP 転送処理のうち VM が停止している時間の割合

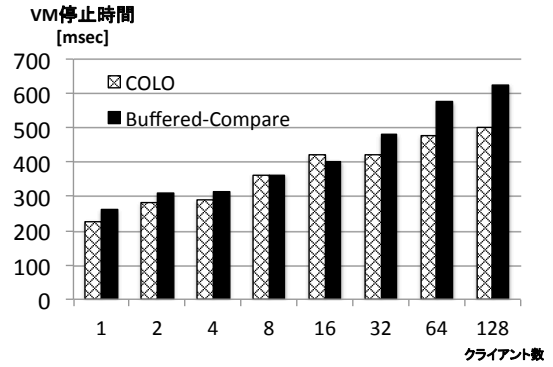

図 11 1 回の CP 転送処理のうち VM が停止している時間

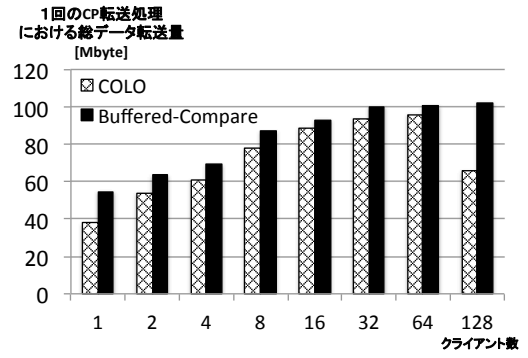

図 12 1 回の CP 転送処理における総データ転送量

Buffered-Compare の効果が現れやすい同時接続クライアント数が 32 以上の場合においては，レスポンスタイムは減少傾向にある．これは，Buffered-Compare の NW 出力の再比較によるレイテンシが増加する影響よりも，Buffered-Compare による CP 転送処理の実施頻度の削減による処理の効率化の方が大きく，レスポンスタイムが短くなったと考えられる．

5.6 評価実験結果:VM 停止時間とデータ転送量

本評価実験では，Buffered-Compare によって CP 転送処理の実施頻度を削減することができることは先の評価で示した．しかし，実施頻度が削減される分，COLO と比較し

1 回の CP 転送処理に負荷がかかってしまう懸念がある．

図 10 は，1 回の CP 転送処理において，VM が停止している時間の割合を示したものである．CP 転送処理では，Pri-VM の状態を Sec-VM へ転送するため，Pri-VM を一時停止状態とし，メモリ差分を転送する準備をした後に Sec-VM へデータ転送を行い，各 VM を再開させるという処理を含んでいる．COLO，Buffered-Compare に関わらず，CP 転送処理において VM が停止している時間が殆どを占めていることがわかる．つまり，CP 転送処理では VM はほぼ停止状態であることがわかる．

図 11 に，1 回の CP 転送処理のうち VM が停止している時間を示す．各同時接続クライアント数において，Buffered-Compare は COLO に比べて VM 停止時間は僅かに長い．これは，Buffered-Compare によって CP 転送処理の実施頻度が削減されたため，1 回の CP 転送処理において COLO に比べてより多くのメモリ差分を Sec-VM へ転送する必要があるためである．事実，1 回の CP 転送処理における総データ転送量を示した図 12 では，COLO に比べて Buffered-Compare でのデータ転送量が増加している．図 12 において，同時接続クライアント数が 128 の場合に COLO のデータ転送量が減少している理由は，CP 転送処理の頻発によって CP 転送処理が行われるまでに発生するメモリ差分が小さくなったためだと考えられる．

この評価から，Buffered-Compare は，COLO に比べて 1 回の CP 転送処理に対する VM 停止時間やデータ転送量は増加傾向であることが確認された．しかし，本評価実験では，これらの影響は Buffered-Compare による CP 転送の実施頻度削減の効果に比べると小さいことがわかった．

6. まとめと今後の課題

本論文では，VMM で VM の冗長構成を維持する方式の中で高効率であるとされている COLO という方式をベースに，VM 上のアプリケーションとして近年一般的に利用されるアプリケーションのように，マルチプロセスで処理を行った際にも，低オーバヘッドで冗長構成を行うことができる一方式として Buffered-Compare を提案した．Buffered-Compare では，VM 上の OS のスケジューリングの非決定性の影響により，各 VM からの NW 出力順序についてマルチプロセス処理時に特に顕著に入れ替りが発生することに着目し，COLO の NW 出力比較アルゴリズムで不一致判定された NW 出力について，設定した閾値の範囲内で再比較を行うアルゴリズムを追加した．これによって，各 VM からの NW 出力順序の入れ替りを考慮し，CP 転送処理の削減が可能となった．本評価実験の結果から，Buffered-Compare は COLO に比べて低オーバヘッドで冗長構成維持ができる場合があることを示した．また，本評価実験では，Buffered-Compare によるレイテンシの増加や 1 回の CP 転送処理の負荷の増加に対する懸念についても評価を行ったが，Buffered-Compare による CP 転送処理の削減効果はこれらの懸念に比べて大きいことを確認できた．

本論文では DBMS の 1 例での評価を実施したが，VM 上で利用されるアプリケーションは多岐にわたる．本提案方式の特性等を判断するため，耐故障技術の適用が必要な様々なワークロードで評価を実施をすることが今後の課題である．また，今回の評価では，Buffered-Compare 内における再比較を繰り返す閾値はヒューリスティックに設定した．今後は，ワークロードの特性によって動的に変化させる等の閾値の最適化についても考察していきたい．Buffered-Compare は，CP 転送処理の実施頻度の削減を実現するための一方式である．冗長構成維持の更なる低オーバヘッド化に向けては，1 回の CP 転送処理に係る処理負荷の削減についても取り組む必要がある．具体的には，CP 転送処理における VM の停止時間の削減や，データ転送の効率化などが考えられる．これらの取り組みについても Buffered-Compare に取り入れ，より低オーバヘッドな冗長構成維持を目指したい．

参考文献

[1] Brendan Cully et al.: Remus : High Availability via Asynchronous Virtual Machine Replication, NSDI 2008
[2] 田村芳明他：Kemari :VM 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 コンピューティングシステム，Vol.3，No.1，13-24(Mar, 2010)
[3] 大村 圭他： KVM を利用した耐故障クラスタリング技術の開発，情報処理学会技術報告，Vol.2010-OS-115，No.20(Aug, 2010)
[4] 大村 圭他：I/O エミュレーションのロギングリプレイによる VM 同期機構の高速化，情報処理学会技術報告，Vol.2011-ARC-195，No.16(Apr, 2011)
[5] Yaozu Dong et al.: COLO: COarse-grained LOck-stepping Virtual Machines for Non-stop Service, 4th ACM Symposium on Cloud Computing(SoCC2013), Oct，2013
[6] 笠江 優美子他：仮想マシン間の低オーバヘッドな冗長構成維持手法実現に向けた既存方式の特性評価，第 132 回システムソフトウェアとオペレーティング・システム研究会，2015-OS-132(2)，1-8(Feb, 2015)
[7] HP NonStopServer : http://h50146.www5.hp.com/products/servers/nonstop/
[8] Stratus ftServer : http://www.stratus.co.jp/Products/Platforms/ftServerSystems
[9] KVM : http://www.linuxkvm.org/page/Main_Page
[10] QEMU : http://wiki.qemu.org/Main_Page
[11] COarse-grain LOck-stepping(COLO) : http://wiki.qemu.org/Features/COLO
[12] colo-v.1.2-basic : https://github.com/coloft/qemu/tree/colo-v1.2-basic
[13] PostgreSQL : http://www.postgresql.org
[14] pgbench : http://www.postgresql.jp/document/9.2/html/pgbench.html
[15] TPC-B : http://www.tpc.org/tpcb/
[16] Al Gillen et al. : Gary ChenWorldwide Virtual Machine 2013-2017 Forecast : Virtualization Buildout Continues Strong IDC , #242762 , Aug 2013