Vol.2010-ARC-188 No.4
2010/3/1

Xenを用いた高可用仮想マシンの検討と試作

庄野篤司[†1]　木村哲郎[†1]

仮想マシンの二重化による高可用化について，実現方法の検討とその妥当性を検証するための機能試作をXenをベースに行った．実現方法として，二つの仮想マシンの入力とその割り込みのタイミングを一致させることで二重化を実現する方式を考案した．加えて，Xenの備えるマイグレーション機能を活用したフェイルオーバーの実現方式を考案した．次に，考案した実現方式の妥当性を確認するべく機能試作を行い，その妥当性を検証した．

Study and Implementation of High Availability Virtual Machine using Xen

Atsushi Shono[†1] and Tetsuro Kimura[†1]

We discussed how to materialize high availability by duplication of virtual machines, and implemented a prototype based on Xen in order to verify its validity. As a way of implementation, we devised a method of duplication of virtual machines by synchronization of input data and its timing to the virtual machines. Additionally, we discussed how to do a failover when failures occur by taking advantage of a migration function which is inherent in Xen. And also, implemented a prototype in order to verify its validity.

1. はじめに

メインフレームの時代から使われてきた仮想マシン技術が，性能が著しく向上した近年のIAサーバ上で再び脚光を浴びている．仮想マシン技術とは，これまで実機のハードウェア環境上で直接稼動していたOSを，実機とは切り離された論理的な環境上で稼動させる技術であり，米国VMware社のVMware ESX Server[1)–2)]やケンブリッジ大学の研究プロジェクトの研究成果がオープンソース化されたXen[3)]などが有名である．

仮想マシン技術の進歩により，1台のサーバ上に複数のOSを稼動させたり，稼働中のOSを止めることなく他のサーバ上に移すマイグレーションが可能となった．これにより，急増してしまったサーバ台数を高性能なサーバに集約するサーバコンソリデーションの実現や，止めることのできないサービスを実行しているサーバのハードウェアのメンテナンスを，マイグレーション機能を活用して可能にしたり，サービス負荷の増減にあわせて負荷分散をマイグレーション機能で実現するなど，サーバ運用に大きな恩恵をもたらしている．

一方，サーバ計算機のハードウェアに生じる故障を隠蔽しサービスを継続する技術として，主要なハードウェアをすべて二重化したフォールトトレラントサーバがある．二重化を制御する専用ハードウェアが搭載されているため，サーバ内部で二重化されたハードウェアは完全に同期実行を行い，ハードウェアに障害が発生した場合に自動的に障害箇所を切り離してサービスを継続する．代表的な製品として，米国Stratus社やNECからフォールトトレラントサーバ[4)5)]がある．

近年の仮想化技術の急速な発展と普及に伴い，専用ハードウェアを用いず仮想マシン技術を応用することで2台の安価なIAサーバ上で同様の耐故障性の実現を目指す技術が台頭してきた．その一つとして，kemari[6)]が挙げられる．kemariでは，I/Oがある度に稼動系から待機系にマイグレーション技術を応用してメモリイメージを転送をする．それにより，もし運用系の仮想マシンが故障しても，待機系が起動して処理を引き継ぐことで，kemariのシステムの外部からは透過的にフェイルオーバが行われる．

我々は，kemariとは異なるアプローチで耐故障性の実現を目指して研究を行ってきた．この高可用仮想マシンは，2台の独立した計算機上のそれぞれの仮想マシン間で入力事象，即ち，外部からの入力と割り込みのタイミングを合わせこむことで，命令レベルで同一の実行状態を保つというものである．

以下の節では，2節で高可用仮想マシンの概要を述べた後，3節で高可用仮想マシンの最も重要な運用系と待機系の同期実行の詳細について述べる．続いて，4節で同期実行機構の試作とその結果について述べ，5節で機能試作で明らかになった同期実行の課題とその解決策について述べる．最後に，6節で結論を述べる．

2. 高可用仮想マシンの概要

我々が目指している高可用仮想マシンは以下の3つの要素技術から構成されている．

†1 東芝研究開発センター
TOSHIBA Research & Development Center

Vol.2010-ARC-188 No.4
2010/3/1

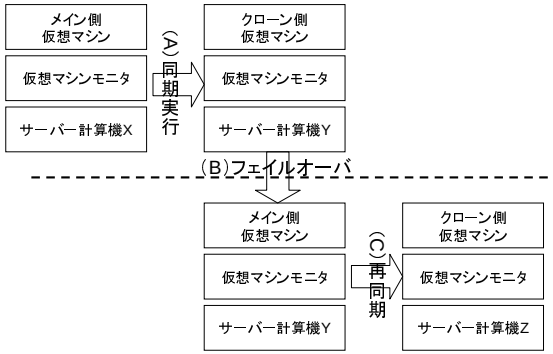

図 1 高可用仮想マシンの概要

- 同期実行（**図 1** の（A））
- フェイルオーバ（図 1 の（B））
- 再同期（図 1 の（C））

以下の節ではそれぞれに対する概要を説明する．

2.1 同期実行の概要

我々が検討した高可用仮想マシンの概要を図 1 の（A）に示す．高可用仮想マシンは，ネットワークで接続された 2 台の独立したサーバで仮想マシンモニタを動作させ，その 2 つの仮想マシンモニタが連携することで，その上で動作する仮想マシンの実行状態を同期させて二重化を図る．独立した 2 台のサーバ計算機上でそれぞれ動作する仮想マシンは，外界からの作用がなければ，同じ初期状態から実行を始めれば，同じ状態遷移をとると考えられる．もし，外界からの作用をも完全にあわせることが出来れば，仮想マシン間で実行状態を同期し続けることが可能となる．

仮想マシンモニタは，仮想マシンに対する外界からの作用，即ち入力事象をすべて制御することが出来る．そのため，仮想マシンへの入力事象を仮想マシンモニタ層で制御することで，2 台の計算機上の仮想マシンモニタが連携してその上で稼動する仮想マシンへの入力事象をあわせこみ，実行状態の同期が可能となる．そこで，2 台の計算機の一方で稼動する仮想マシンの実行状態を他方で稼動する仮想マシンにコピーするという二重化を考える．ここで，前者をメイン，後者をクローンと呼ぶことにする．

高可用仮想マシンでは，メイン側仮想マシンへの入力は，コピーした上でクローン側仮想マシンに対する入力とする（**図 2**）．一方，出力に関しては，メイン側とクローン側仮想マシンが出す出力要求は，両方の仮想マシンが同じ実行状態を維持していれば，同じ出力要求が出されるはずである．つまり，同期実行の実現には，出力は特別の処理は基本的には不要である．しかし，仮想マシンから仮想マシンモニタに渡された出力要求の扱いは，メイン側とクローン側で異なる．メイン側では出力要求はそのまま通常の仮想マシンと同じく物理デバイスに反映させる．一方，クローン側では仮想マシンモニタで出力要求は破棄される（**図 3**）．これは，高可用仮想マシンからの出力要求の重複を回避するためである．

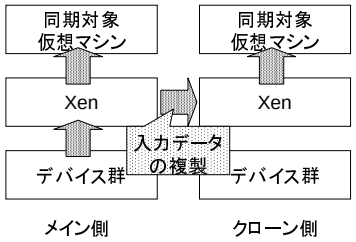

図 2 入力データの流れ

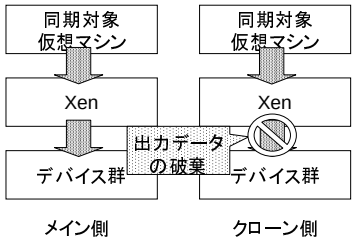

図 3 出力データの流れ

2.2 フェイルオーバの概要

高可用仮想マシンでは，クローン側の実行に必要な入力事象はメイン側の値を複製して使っている．このため，同期実行を行っている間は，クローン側は実計算機とは切り離された状態で動作している．故に，メイン側計算機の障害発生に起因してフェイルオーバを行い，クローン側仮想マシンを自立運用させるためには，実計算機への適合処理が必要となる．

一方，Xen に備わっているマイグレーション機能は，移動元の計算機から仮想マシンを切り離し，移動先で動作させる．そのため，移動先で動作を開始するにあたって，実計算機への適合処理が必要となる．

上述のようにフェイルオーバとマイグレーションには共に実計算機への適合処理が必要とされる．その類似性から，フェイルオーバを行う際は，クローン側仮想マシンをローカルホストへマイグレーションを行なうことで，実計算機への適合処理の大部分を流用でき，試作コストの大幅な低減が期待できる．ここで，実計算機への適合処理とは，Xen で仮想化された環境で稼動する高可用仮想マシンの場合，実計算機とインタラクションを行う特権ドメインとの結合を指す．具体的には，割り込みの通知機構であるイベントチャネルの結合や，I/O データを置くバッファの生成と共有等である．

2.3 再同期の概要

フェイルオーバが実行され，クローン側が単体で稼動する状態になると，一時的に非二重化状態での運用となる．そこで，前述のマイグレーション技術を応用することで，単体で動作している仮想マシンのイメージを転送し新たなクローンとすることで非二重化状態を解消させる．これは，高可用仮想マシンが特定の 2 台のハードウェアを固定的に組み合わせて二重化を実現するのではなく，その時々で利用可能な 2 台を組み合わせれば良いという，高可用仮想マシンの構成の自由度の高さを示している．

本稿の以降の節では，高可用仮想マシンを実現する上で最も重要な同期実行機構に絞って，二重化の実現方式や機能試作などについて示す．

3. 同期実行の詳細

我々が考えた同期方法は，仮想マシンに対する外部環境からの作用を同じにすれば，2 つの仮想マシンは同じ状態遷移をするであろうという推測に基づく同期方法である．この方法では，外部環境からの作用の一致を仮想マシンモニタで保証し，決定的な状態遷移をするメモリアクセスを行っている場合は同期操作をせずに通常の処理速度で仮想マシンを実行する．この方法により，同期処理の頻度と負荷が軽減され，同期に伴うオーバーヘッドを大幅に低減できる．

3.1 Xen の入力事象処理機構

図 4 に Xen の入力事象の処理機構の概要を示す．図は実デバイスからの入力データを割り込みを利用して管理用の仮想マシン dom0 から一般用の仮想マシン domU まで受け渡しが行われる様子を示している．dom0 とは異なり，domU は実デバイスドライバは持たず，仮想化された仮想デバイスドライバのみを持つ．一方，dom0 とは，実デバイスからの実割り込みと入力データを受け付け，それが domU へのものならイベントチャネル（図中 evtchn）を介して domU に通知するとともに，domU との共有メモリを介して入力データを受け渡す役割を担う．

以下に，domU への入力データがハードウェアから domU に受け渡されるまでの処理フローの概要を示す．

(1) dom0 がデバイスドライバからの実割り込みを受け付け入力データを読み込む
(2) dom0 の仮想デバイスドライバは読み込んだ入力データを domU との共有メモリ上に置く
(3) dom0 の仮想デバイスドライバはイベントチャネルに通知を出す（イベントチャネルの対応するビットを立てる）
(4) Xen は制御が domU に移る直前にイベントチャネルのビットが立っていれば対応する仮想割り込みに変換し domU に通知する
(5) domU の仮想デバイスドライバは仮想割り込みを受け付ける
(6) domU の仮想デバイスドライバは共有メモリから入力データを読み出す

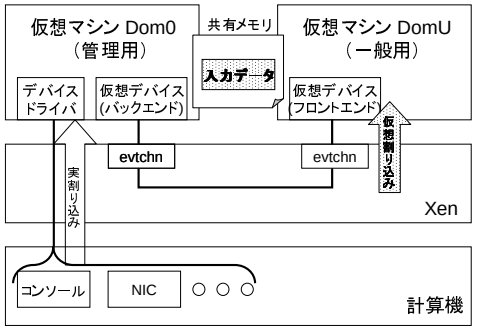

図 4 Xen の入力事象処理機構

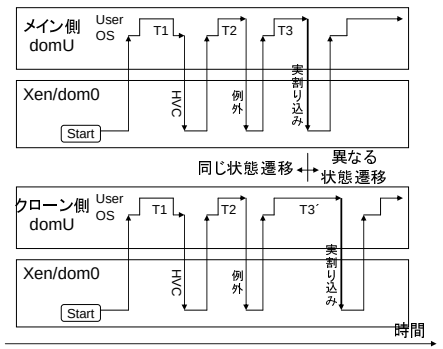

図 5 高可用仮想マシンの状態遷移（1）

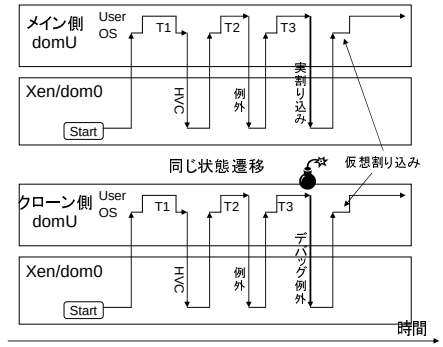

図 6 高可用仮想マシンの状態遷移（2）

3.2 Xen と domU の実行状態遷移のタイミング

図 5 と**図 6** に同期実行するメインとクローンの状態遷移を示す．図中，メイン側 domU とクローン側 domU が同期対象ドメインである．図では，横線で domU または dom0/Xen のどれが実行されているかを示している．また，T1 や T2 はメインとクローンが同期して

実行している期間（以後、タームと呼ぶ）を表している．T1 はハイパーバイザコール（以後，HVC と略す）でタームが終わった場合を，T2 は例外でタームが終わった場合を示している．ここで，図 5 では T1, T2 は同期して実行しているが，T3 で同期が取れなくなってしまった様子を示している．これは 3.2.2 節で示す実割り込みの同期機構が導入されていないためである．一方，図 6 では同機構を取り入れたために，メイン側の T3 と同じタイミングでクローン側の T3 で擬似的な実割り込みを実現した様子を示している．同機構ではメイン側で実割り込みで制御が Xen に移ったタイミングをクローン側でブレークポイントを用いて強制的に同じタイミングで Xen に制御を移すことで T3 の一致を図る．より詳しくは，3.2.2 節を参照のこと．

同期実行を実現するためには，入力事象の合わせこみ，即ち入力データと仮想割り込みのタイミングを合わせこむ必要がある．仮想割り込みが発生するのは，domU から Xen に制御が移り，再度 domU に制御が移るタイミングで，イベントチャネルのビットが立っている場合である．故に，メインとクローンの domU が実行する命令列中の同じ位置で domU から Xen に制御が移る必要がある．Xen に制御が移る要因には，例外と HVC，実割り込みの 3 種類がある．次節でそれぞれについてみていく．

3.2.1 例外とハイパーバイザコール

仮想マシンの実行中に発生する例外は，仮想マシンの処理内容や内部状態に起因して発生するイベントであり，仮想マシンの状態遷移の中で決定的に発生する[⋆1]．また，HVC も仮想マシンが実行する命令列中に実行命令があるため，仮想マシンの状態遷移の中で決定的に発生する．そのため，これらを契機として制御が仮想マシンから仮想マシンモニタに移った場合，タイミングの同期のための特別な処理は不要である．

3.2.2 実割り込みとそのタイミングの一致

実割り込みは CPU が実行する命令列とは独立に発生するため，非決定的な状態遷移の大きな要因である．そこで，我々は実割り込みのタイミングを擬似的に合わせる手法を検討した．この手法では，メイン側 domU は普通に処理を実行し割り込みを Xen 経由で受け取るが，クローン側 domU はメイン側 domU の割り込み位置と同じ場所で処理を中断させて Xen に制御を移すことにより，擬似的な実割り込みを再現する．この方法では，メイン側の割り込み位置をクローン側に伝えて，クローン側で対応する擬似実割り込みにより制御を奪うということを行うため，クローン側の処理がメイン側より遅れて進むことが前提である．

擬似実割り込みの実現方法としては，メイン側 domU の実割り込みが起きた位置情報（プログラムカウンタ）を元にクローン側 domU にブレークポイントを設定することでデバッグ例外を発生させ，実行を中断する方法が考えられる（図 6 の爆弾印）．しかし，割り込み位置がループ中にあった場合，ループの何周目で割り込みが掛かったかが分からないため，正確な割り込みの再現ができない．そこで，パフォーマンスカウンタと呼ばれる，実行した命令数をカウントするレジスタを利用する方法を検討した．このレジスタは Intel®の Pentium®プロセッサ[⋆2]に搭載されている性能評価用の機能の一部である．この機能を利用することで，メイン側の仮想マシンで前回の割り込みから今回の割り込みまでの間に実行した命令数をカウントし，クローン側ではブレークポイントで処理が中断されるたびに実行命令数を比較することで，ループ中であっても正確に擬似割り込みを再現可能となる．

3.3 入力データと仮想割り込み

仮想マシンに対する外部環境からの作用としては，仮想割り込みとそれに伴うデータ入力が挙げられる．本節では，これらの作用に対する同期制御方法について説明する．

3.3.1 仮想デバイスからの入力データの一致

仮想マシンが仮想デバイスから読み込むデータは，入力データとして仮想マシンの実行状態の遷移に大きな影響を与える．そのため，2 つの仮想マシンが仮想デバイスから読み込むデータの内容を一致させる必要がある．

本来，各サーバ計算機上で稼動する仮想マシンが持つ仮想デバイスの多くは，そのサーバ計算機の物理デバイスを仮想化したものであるため，異なるサーバ計算機上で稼動する 2 つの仮想マシンから見える仮想デバイスの状態は異なる．独立したハードウェアである各サーバ計算機の物理デバイスの状態を一致させるのは非常に困難なため，高可用仮想マシンでは片方のサーバ計算機の物理デバイスの状態を元に，仮想デバイスの状態を決定し 2 つの仮想マシンの仮想デバイスに反映する方法をとった．即ち，メイン側への入力データを複製し，クローン側の仮想マシンの入力データとする方法である（図 2）．

3.3.2 仮想割り込みの一致

仮想割り込みは，Xen から domU に制御が移ったタイミングで，イベントチャネルへの通知があったことを示すビットが立っていれば，domU に配送される．そこで，実割り込みや HVC，例外で Xen に制御が移り，再度 domU に制御が移るタイミングでイベントチャネルのビットを調べる．もしビットが立っていれば，入力データと仮想割り込みの種類，プ

⋆1 5.4 節に示す例外は非決定的に発生する

⋆2 Intel, Pentium, Celeron は，アメリカ合衆国およびその他の国における Intel Corporation の商標です．

表 1　機能試作の環境

プロセッサ	Celeron®-D
メモリ容量	1GB
ハイパーバイザ	Xen-3.0.3
仮想マシンの OS	Linux 2.6.19
仮想マシンのメモリ容量	192MB

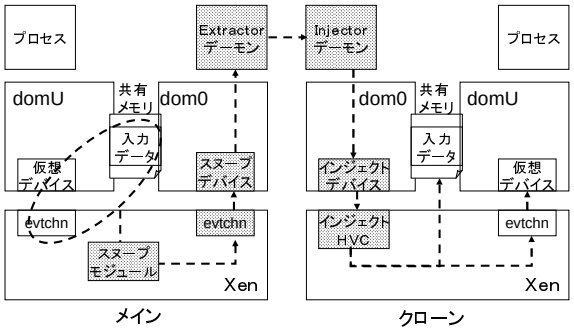

図 7　同期実行機構の基本的なアーキテクチャ

ログラムカウンタの値，タームでの実行命令数等を取得し，クローン側に送る．クローン側では，送られてきた諸情報を元にメイン側と同じタイミングで入力データを共有メモリ上に置くとともに，イベントチャネルの対応するビットをたて仮想割り込みを発生させる．これによりメインとクローンで同じタイミングで入力事象を同期させることが出来る．

4. 同期実行機構の機能試作

3 節に示した同期実行機構の機能試作を行った．機能試作を行った環境について**表 1** に示す．メイン側とクローン側の双方で同環境にて機能試作を行った．また，**図 7** に同期実行を実現するための基本アーキテクチャを示す．試作に当たって新たに追加した機能ブロックを網掛けで示している．

メイン側 Xen のスヌープモジュールは，イベントチャネル (domU への仮想割り込み) を監視し，domU に制御が移る際にもしイベントチャネルのビットが立っていれば，共有メモリに置かれた入力データとイベントチャネルの情報を Xen 内に設けたバッファに取得（以後，スヌープと呼ぶ）複製し，新たに設けたイベントチャネルを介して dom0 に通知する．通知を受けた dom0 のスヌープデバイスはスヌープしたデータを Xen から取得し，Extractor デーモンを介してクローン側に送信する．クローン側で動作する Injector デーモンはメイン側からデータを受信すると，インジェクトデバイスにデータを受け渡す．インジェクトデバイスはクローン側 Xen に設けたインジェクトハイパーバイザコールを呼び出し，Xen にデータを受け渡す．インジェクトハイパーバイザコールは共有メモリに入力データを挿入するとともに，イベントチャネルの対応するビットを立て通知をだす（以後，インジェクションと呼ぶ）．クローン側の domU は仮想割り込みを受信し，共有メモリ上に置かれた入力データを読み出す．これにより，メイン側でスヌープした入力データとイベントチャネルをクローン側でインジェクションし，同期実行を実現する．

上記の機能試作により，ブートから数分間同期して起動することを確認した．ただし，未発見な同期実行阻害要因，もしくはバグによって，途中から同期がとれなくなってしまう現象が観測されている．同要因またはバグの解決を図る必要がある．また，同期速度を評価するために，ブートからログインプロンプトが表示されるまでの時間を計測した．その結果，ブートからログインプロンプトが表示されるまでにかかる時間は，高可用仮想マシンのための改造を施す前の約 3 倍かかることが判明した．これに関しては，入力データをスヌープして Extractor デーモンに渡すまでに行っているメモリコピーの回数を削減することで性能向上が期待できる．

5. 同期実行の課題とその解決方法

本節では，機能試作を行い，同期実行の実現を試みた際に明らかになった同期実行を阻害する問題点について，主なものの内容と解決方法について示す．

5.1 Time Stamp Counter

高可用仮想マシンでは，実装プラットフォームとして Xen をベースにしている．同期実行の基本コンセプトは，外界からのすべての入力事象を Xen が仲介し，そこでメイン側仮想マシンとクローン側仮想マシンの間でコピーを行う，というものであったが，機能試作を行う過程で，Xen が仮想化していない外界情報があることが判明した．

それは，プロセッサ内部にある Time Stamp Counter(TSC) と呼ばれるレジスタの参照である．この TSC は，プロセッサに供給されたクロックを数えるレジスタであり，仮想マシンのとしての Linux は時刻補正に TSC の値を読み出す命令 (rdtsc) を直接発行しており，Xen はそれを仲介していない．そのため，Xen 内部の機能拡張だけでは対処できないため，

Linux 内部の rdtsc 命令周辺に改造を加えることとした．方法としては以下の二通りを検討した．

(1) **rdtsc を HVC に置換する**

TSC を読み出す HVC を新設して rdtsc 命令を置換する．HVC を介した読み出しであれば，読み出される TSC の値を Xen で捕捉してクローン側に伝達することが簡単に実現できる．この方法の欠点は，コストの大きい HVC への置き換えでオーバーヘッドの増大を招くところである．

(2) **rdtsc のログを取る**

メイン側 domU で読み出した TSC の値のログを domU と Xen の共有メモリに保存し，タームの終了時に Xen がログを読み出す方法がある．この方法では，ログ格納領域への TSC のコピーだけのコスト増で抑えられるため，オーバーヘッドの増加を非常に低く抑えられる．注意点としては，メイン側 domU とクローン側 domU がまったく同じ命令列を実行しつつ，メイン側ではログの保存，クローン側ではログの読み出しを行わなければならない点である．そこで，以下に示す手法を検討した．

Xen と domU の共有メモリ上に TSC の値を格納するリングバッファとその読み込み用のインデックスと書き込み用のインデックスの二つを設ける．メイン側ではいずれのインデックスも 1 に，クローン側では読み込み用と書き込み用のインデックスをそれぞれ 1 と 0 にそれぞれの Xen が初期化する．メイン側の domU は，

(a) rdtsc 命令を発行
(b) 値を書き込み用のインデックスでリングバッファに書き込む
(c) 値を読み込み用のインデックスでリングバッファから読み出す
(d) 二つのインデックスをともに 1 インクリメントする

を実行する．制御が Xen に移ったら，Xen はリングバッファから値を取得し，同期情報としてクローン側に送る．一方，クローン側の Xen は同期情報から取得した TSC の値をリングバッファに書き込み，クローン側 domU は（a）〜（d）の処理を行う．このように，rdtsc 命令を（a）〜（d）で置き換えることで，メイン側ではログの書き込み，クローン側ではログの読み込みを行うことができる．

オーバーヘッドを考慮した結果，機能試作では（2）の方法を採用した．

5.2 出力用バッファ

Xen では，domU の出力データは dom0 との共有メモリ上のバッファに格納され，実際のデバイスに対する入出力を代行する dom0 に受け渡される．通常は，dom0 がバッファ中のデータを全て受理してから domU に制御が移るため，domU が新たに出力できるデータはバッファサイズ分だけ連続して発行できる．ところが，dom0 の処理が重くなったりスケジューラによるスケジューリングにより dom0 がバッファ中の出力データを全て受理する前に domU に制御が移ると，新たに出せる出力データはバッファの空き領域に依存して変化する．この変化量は同期せずそれぞれ独立に動作するメインとクローンの dom0 に依存するため，同期実行を阻害する原因となる．

高可用仮想マシンでは，dom0 は同時実行を支援するためにメイン側とクローン側で異なる処理を担当しており，出力データの受理を足並みをそろえて実行するのは困難である．そこで，メインとクローンの双方でバッファのデータが全て受理されるまで，domU に制御が移らないようにする方針をとることとした．具体的には，dom0 が出力データ受理を開始する前に domU を pause させ，dom0 が全て受理してバッファが空になった時点で domU を unpause させることとした．

5.3 RF ビット

システムの状態管理を行うために IA-32 に導入されている EFLAGS システムレジスタの RF(Resume Flag) ビットは，立っている場合デバッグ例外の発生を抑制する．一般的には，rep プレフィックス命令[⋆1]を実行中に割り込みが発生すると立つフラグである．フラグが立っているとデバッグ例外の発生が抑制されるため，ブレークポイントで制御を Xen に移すことができず，擬似的な実割り込みの再現ができなくなる．

そこで，もし実割り込みで Xen に制御が移った場合，RF ビットが立っているかを判定し，もし立っていれば仮想割り込みの発生を次のタームまで遅延させるという対策をとった．

5.4 Device Not Available 例外

浮動小数点演算レジスタの切り替えに用いられる Device Not Available 例外は，それ以前に実行していたドメインに依存して非決定的に発生する．即ち，Device Not Available 例外の発生はドメインスケジューリングに依存している．故に，メイン側が Device Not Available 例外をタームの終わりとすると，クローン側の命令ストリーム中の同じ位置 (浮動小数点演算命令) で同例外が発生するとは限らず，同期実行を阻害してしまう．

そこで，もし Device Not Available で Xen に制御が移った場合，RF ビットの場合と同じく仮想割り込みの発生を次のタームまで遅延させるという対策をとった．

⋆1 ストリング命令などのプレフィックス命令であり，繰り返し処理の記述の簡易化に用いられる

Vol.2010-ARC-188 No.4
2010/3/1

6. おわりに

本稿では，2 台の独立した計算機上で実行される仮想マシン間で，入力データとそのタイミングを一致させることで，仮想マシンレベルでの二重化を実現する方式に関して示した．また，マイグレーションを利用したフェイルオーバ機構に関してもその概要を示した．

同期機構はまだ一部に不安定なところがあるものの基本的な同期実行が正常に動作することを確認した．また，フェイルオーバ機構も正常に動作し，障害発生時にクローンが単独でサービスを継続できることを確認した．

今後は，同期動作の安定性と速度の向上を図るとともに，再同期機構の設計と実装を行うことで，高可用仮想マシンの実現を目指す．

参 考 文 献

1) VMware ESX Server. http://www.vmware.com/products/vi/esx/.
2) Adams, K. and Agesen, O.: A Comparison of Software and Hardware Techniques for x86 Virtualization, *SIGARCH Comput. Archit. News*, Vol. 34, No. 5, pp. 2–13 (2006).
3) Xen. http://xen.org.
4) Stratus ftServer Family. http://www.stratus.com/products/ftserer/index.htm.
5) NEC ft サーバ. http://ww.express.nec.co.jp/products/pcserver/ft/tokucho.shtml.
6) Kemari. http://www.osrg.net/kemari/.