社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－OS－105 (10)
2007／4／5

仮想マシン間の同期による高可用クラスタリング方式の提案

田村芳明† 佐藤孝治†
木原誠司† 盛合敏†

インターネット上で提供されるサービスの増加と高機能化に加え，PCサーバの小型化，高速化，低価格化により，企業では多数のPCサーバで構成された複雑なシステムのコスト削減とリソースの有効利用が求められている．この課題を解決するために，仮想マシンを利用して，1つの物理マシン上に複数のサーバ機能を統合することが検討されている．しかし，ハードウェア障害発生時にサービスを継続するためには，特殊なハードウェア，アプリケーションやOSに依存しない，可用性の高い構成が必要である．本稿ではアプリケーションやOSに依存しないで障害発生時にサービスを継続する，仮想マシン間の同期によるクラスタリング技術を提案する．さらに，仮想マシンモニタであるXenを利用したプロトタイプの実装と評価を行い，提案方式の有効性を確認した．

Virtual Machine Synchronization for High Availability Clusters

Yoshiaki Tamura,† Koji Sato,† Seiji Kihara† and Satoshi Moriai †

Internet services are growing in numbers and functionality. Commodity servers are well used for those services, and service providers face a problem to lower the cost of running numbers of servers. Consolidating multiple servers by using virtual machines is an effective approach to reduce unnecessary costs such as floor space, power, and management, which results in better resource utilization. On the other hand, consolidating multiple servers to a single server may increase the risk of single point of failure. Although high availability architectures should be considered for this, current solutions require specially designed hardware, or modifications to running software. In this paper we propose a cluster system that synchronizes virtual machines for high availability, which does not require specific hardware or modifications to software. We present the design, implementation and evaluation of our approach.

1. はじめに

近年，インターネットの普及により，広告やショッピング，医療など様々なサービスがインターネット上で提供され，その数も増加している．また，インターネットサービスのインフラストラクチャに広く用いられているPCサーバは，技術の進歩と共にコモディティ化が進んでいる．PCサーバは手軽に導入できるため，企業で使われるサービスにも多く利用されている．しかし今日の企業では，多数のPCサーバで構成される複雑なシステムのコスト削減とリソースの有効利用が課題となっている．この課題を解決する方法として，仮想マシンを利用したサーバ統合が多くの企業で検討されている．サーバ機能は仮想マシンによって提供され，複数の仮想マシンが1つの物理マシン上で動作することになる．これにより，設置スペースの削減，消費電力の削減，管理対象となるハードウェアの削減やリソース利用率の向上が可能になると考えられている．

しかし，仮想マシンを使ったサーバ統合には問題もある．サーバを統合する前のシステムの場合，1つのサーバにハードウェア障害が発生したとしても，他のサーバに影響は無い．一方，サーバ統合環境では，統合先のサーバにハードウェア障害が発生した場合，故障したサーバで動作する全ての仮想マシンに影響が広がる．このためサーバ統合環境では，サーバ統合をする前よりも多くのサービスが停止するという問題があり，これを解決するためには高可用性が必要になる．

従来の高可用化技術としては，フォールトトレラントサーバ（FTサーバ），高可用クラスタ（HAクラスタ）のアクティブスタンバイ方式やレプリケーション方式が挙げられる．FTサーバではソフトウェアを変更する必要はないが，ハードウェアのコストが大きい．HAクラスタのアクティブスタンバイ方式では障害発生後の切替に時間を要する．また，レプリケーション方式ではソフトウェアの変更を必要とするため，サー

† 日本電信電話株式会社 サイバースペース研究所
NTT Cyber Space Laboratories

バ統合環境に適用すると，仮想マシン上の全てのソフトウェアに変更を加えなければならない．

本稿では，安価なPCサーバを対象に，アプリケーションやOSに依存しないで，障害発生時にサービスを継続する技術を提案する．提案技術では，仮想マシン間の同期を取ることで，アプリケーションやOSに依存せずに，サービスを継続することができる．

2. 既存のクラスタリング技術と問題点

2.1 FTサーバ

FTサーバでは，CPU，メモリ，ストレージ，ネットワーク，電源などの主要なハードウェアコンポーネントが冗長化されている．各コンポーネントは，運用系の状態が変化すると，待機系の状態を運用系と一致させる処理が行われる．ここで，各コンポーネントに対する入力によって，状態が変化することを状態遷移という．ここでは，状態を一致させる処理を同期と呼ぶ．FTサーバでは，運用系が1命令進むごとに，待機系に運用系と同じ状態遷移をさせることで同期を行っている．さらに，同期処理をハードウェアで実装することで，高速に同期を行うことができる．アプリケーションやOSが動作するのに必要なコンポーネントは，常に同期されているため，運用系に障害が発生して停止したとしても，待機系が処理を引き続いて行うことができる．従って，アプリケーションやOSが特別な処理を必要としないで，透過的にサービスを継続することができる．しかし，FTサーバは特殊なハードウェアで構成されるため，同程度の性能を持つPCサーバと比較して10倍程度の導入コストがかかる．

2.2 HAクラスタ

HAクラスタのアクティブスタンバイ方式では，運用系と待機系のサーバを用意し，運用系と待機系でストレージを共有する．運用系に障害が発生して停止した場合は待機系に切替える．これをフェイルオーバという．前に挙げたFTサーバと異なり，汎用的なハードウェアを利用することができるため，安価に構築することができる．運用系は待機系を同期するのに必要な，アプリケーションに依存した情報を共有ストレージに書き，待機系はフェイルオーバ時にこの情報を用いてリカバリを行う．このように，アクティブスタンバイ方式では，フェイルオーバ時にアプリケーション毎のリカバリ処理が必要になるため，アプリケーションやOSから見て，透過的に可用性を得ることができない．また，フェイルオーバには時間を要し，その間はサービスが提供できなくなるという問題がある．

HAクラスタのレプリケーション方式も，アクティブスタンバイと同様に，運用系と待機系を用意する．アクティブスタンバイ方式とは異なり，それぞれのサーバが個別にストレージを持つ．レプリケーションの具体的な方法として，運用系のアプリケーションに来たリクエストを待機系にも転送することが挙げられる．運用系に障害が発生して停止した場合，待機系のアプリケーションの状態は運用系と同期されているため，運用系を切り離してサービスを継続することができる．しかし，クラスタリングするアプリケーション毎にレプリケーションの仕組みを追加しなければならないため，アプリケーションやOSから見て，透過的に可用性を得ることができない．

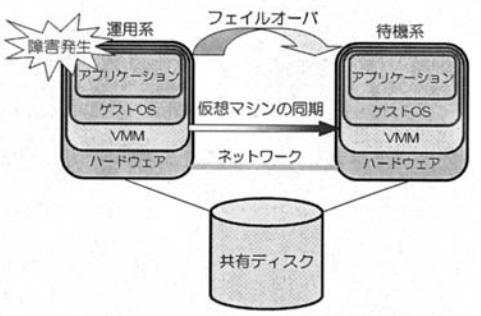

図1 仮想マシン間の同期によるクラスタリング

3. 仮想マシン間の同期よるクラスタリングとその課題

3.1 仮想マシン間の同期によるクラスタリング技術の提案

本稿では，既存のクラスタリング技術の問題を解決するために，仮想マシン間の同期によるクラスタリング技術を提案する(図1)．運用系がどの時点で停止しても，待機系が同じ状態になるように同期を行うことで，アプリケーションやOSに依存せずに，障害発生時にサービスを継続することを可能にする．提案技術は，企業内の情報共有サーバなど，コストと可用性の両立を求められるサービスを対象とする．本技術を実現するためには以下の機能が必要となる．

(1) 仮想マシンの同期
(2) ハードウェア障害の検知
(3) 障害発生後の切り替え

このうち，(2)と(3)については，既存のHAクラスタで同様の機能が実現されているため，それらを仮想マシン間の同期によるクラスタリング技術に応用できる．従って，本稿では(1)の仮想マシンの同期について述べる．

3.2 仮想マシン間の同期における課題

図 2 に，仮想マシン間の同期の流れを示す．待機系の仮想マシンの状態を運用系の仮想マシンと同じにするために，まず運用系の動作を一時的に停止し，運用系と待機系の同期を開始する．続いて，待機系の状態が更新されたことを確認し，運用系を再開する．t_{sync} は同期にかかる時間，$t_{interval}$ は同期を行う間隔を表す．このとき，仮想マシン間の同期にかかるオーバヘッドは，$t_{interval}$ に対する t_{sync} の長さによって決まる．仮想マシン間の同期は，同期の準備，同期に必要なデータの転送，同期の完了待ちによって構成される．このうち，同期に必要なデータの転送が最も時間がかかると考えられる．同期に必要なデータの転送時間を短くするためには，転送するデータを小さくする必要がある．転送データを小さくする方法としては，仮想マシンの CPU やメモリを対象に，前回の同期からの差分だけを転送する手法が挙げられる．

表 1 に，仮想マシンの差分転送を利用し，10ms，100ms，1s，10s の間隔で同期をしたときの 1 回の同期にかかった時間を示す．表 2 の測定環境 2 台を Gigabit Ethernet で接続し，ゲスト OS にメモリを 1GB 割り当てた．ネットワークのベンチマークツールである Netperf[3] を用い，クライアントから 512MB のデータを転送することで，仮想マシンの負荷を発生させた．測定環境のページサイズは 4KB である．同期の間隔を 10 倍ずつ増加させたとしても，同期にかかる時間は約 1.1 倍しか増加しない．アプリケーションにも依存するが，差分転送が有効に機能しているためだと考えられる．

このように，仮想マシン間の同期のオーバヘッドを下げるためには，同期の頻度を少なくし，間隔を長くすることが有効である．しかし，一定の時間間隔で同期を行った場合，運用系と待機系を常に同じ状態に保つことはできないため，アプリケーションや OS に依存せずに，障害発生時にサービスを継続することができない．そこで，運用系の状態遷移が外部とのイベントによって起きることに着目し，これを同期の契機として利用する．

仮想マシンと接続されるデバイスには，タイマ，ネットワーク，ストレージ，コンソールなどがある．本技術が想定している環境では，タイマ，ネットワーク，ストレージのイベントによる同期が中心になり，これらのイベントを契機に同期を行った場合，ネットワークとファイル I/O の性能が特に影響を受けると考えられる．そこで，ベンチマークに Netperf と IOzone[2] を利用し，同期を行った場合のネットワークとファイル I/O の性能を測定した (表 3)．ファイル I/O の性能測定では 512MB のデータを 32KB ずつ，同期書き込みとバッファリング付き書き込みで書き出した．このように，前述のデバイスとのイベントを契機に同期を行った場合，同期のオーバヘッドにより，仮想化のみと比較してネットワークで 24%，ファイル I/O の同期書き込みで 10%，バッファリング付き書き込みで 24%の性能にまで低下することがわかった．

本稿では，同期のオーバヘッドを下げ，最終的に仮想化のみを基準として 50%の性能を得ることを目標とする．これは，2 倍の導入コストで半分の性能が得られれば，10 倍の導入コストがかかる FT サーバと比較しても優位性があるからである．同期のオーバヘッドを下げるには，同期の契機となるイベントを選定し，同期の頻度を削減する必要がある．

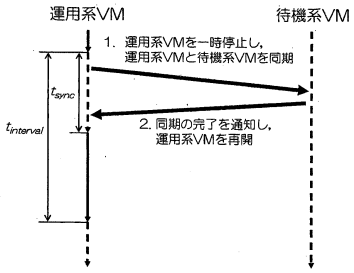

図 2 仮想マシン間の同期

表 1 同期間隔と同期時間

間隔	ページ数	転送時間 [ms]	同期時間 [ms]
10ms	253	7.847	15.704
100ms	282	8.839	16.804
1s	323	11.033	19.147
10s	372	13.244	21.605

表 2 測定環境

サーバ	HP DL360G5
CPU	Intel Xeon 5160 3GHz
メモリ	DDR2-667 4GB
NIC (GbE)	Broadcom NetXtreme II BCM5708
FC アダプタ	QLogic QLA2300
ストレージ	dotHILL SANnet II 200FC
ディスク	Seagate ST3146807FC 147GB
仮想マシンモニタ	Xen unstable
ゲスト OS	Debian Etch Testing

表 3 同期の有無と仮想マシンの性能

	ネットワーク	ファイル I/O [MB/s]	
	[Mbps]	同期書き込み	バッファリング
仮想化のみ	93.39	5.67	53.95
全イベントで同期	22.81	0.58	16.54

4. 同期する頻度の削減方法

3.2 節で述べたように，仮想マシンに関係する外部とのイベント全てを契機に同期を行うと，性能が著しく低下する．本節では，仮想マシンに接続されるデバイスの種類とイベントの方向に着目し，同期の契機として用いるイベントを選択する．

4.1 同期の契機とするイベントの種類

仮想マシンに接続されるデバイスとして，3.2 節で述べた，タイマ，ネットワーク，ストレージ，コンソールを対象として，同期の契機とする必要性について述べる．

まず，タイマのイベントについて説明する．仮想マシンがタイマからイベントを受け取ると，仮想マシンの状態は遷移する．しかし，実時間に依存したアプリケーションでなければ，受け取ったイベント以降の時刻に送られてくるイベントによって，同じ状態遷移を引き起こすことが可能である．提案方式では，タイマのイベントを捕捉しないため，実時間に依存したアプリケーションは対象外となる．

一方，ネットワーク，ストレージ，コンソールのイベントについて説明する．仮想マシンがこれらのイベントを受け取ると，仮想マシンの状態は遷移する．反対に，仮想マシンがこれらのデバイスにイベントを送ると，デバイスの状態は遷移する．ここで起こる状態遷移はやり取りされるイベントによって異なるため，ネットワーク，ストレージ，コンソールとのイベントは捕捉する必要がある．

4.2 同期の契機とするイベントの方向

4.1 節で候補としたイベントのうち，ネットワークとストレージについて，仮想マシンから見た入力と出力のイベントに分け，各イベントを同期の契機とする必要性について述べる．尚，本技術が想定しているサーバ環境ではコンソールのイベントは少ないため，イベントの方向に関する検討は行わない．

4.2.1 ネットワークのイベント

図 3 に，提案する同期方式と，運用系，待機系，通信相手の状態遷移を示す．運用系の仮想マシンがネットワークから受け取るイベントは，通信相手が仮想マシンに送信を行った場合に発生する．このイベントにより，仮想マシンはイベントの情報に基づいた状態遷移を行う．ネットワークから運用系の仮想マシンへのイベントを契機に同期をしなかった場合，待機系はこのイベント以前の状態から再開することになる．しかし，待機系は運用系がネットワークから受け取ったイベントを知らないため，待機系と通信相手の関係は整

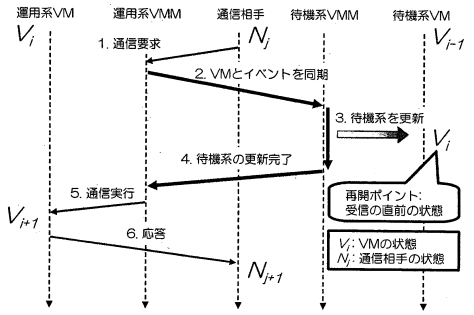

図 3 ネットワークからのイベントによる同期

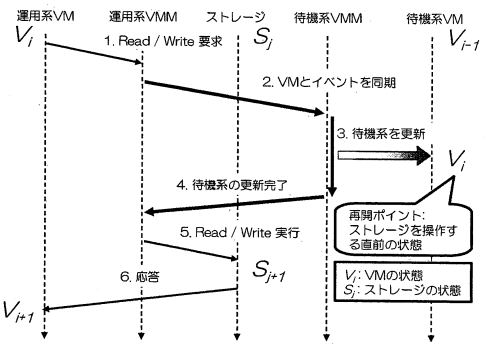

図 4 ストレージへのイベントによる同期

合性の無い状態になってしまう．従って，ネットワークから仮想マシンへのイベントを契機に運用系と待機系の同期を取らなければならない．

一方，仮想マシンからネットワークへのイベントは，仮想マシンから通信相手への送信によって発生する．ネットワークから運用系の仮想マシンへのイベントを契機に同期をした場合，待機系はネットワークから受信する直前の状態から再開する．待機系はネットワークからのイベントを受け取り，運用系と同じ状態遷移を行うため，運用系の仮想マシンからネットワークへのイベントを再現することができる．待機系は，再開時に運用系が送信を完了していたとしても，運用系と同じデータを同じ通信相手に送ることになる．運用系と待機系の二重送信によって，通信のシーケンスに変化は起きるが，これは TCP のような信頼性を保証する通信プロトコルではプロトコルで対処し，UDP のような信頼性を保証しないものではアプリケーションが対処するため，問題は生じない．従って，運用系の仮想マシンからネットワークへのイベントは捕捉する必要がない．

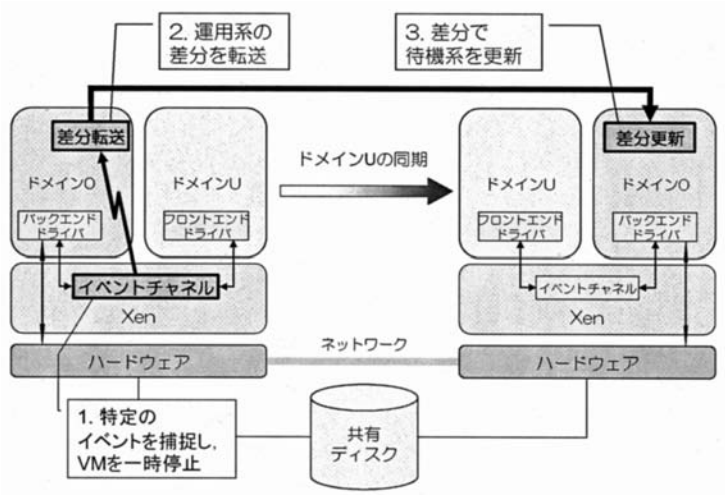

図 5　Xen を利用したプロトタイプの概観

4.2.2　ストレージのイベント

図 4 に，提案する同期方式と，運用系，待機系，ストレージの状態遷移を示す．運用系の仮想マシンからストレージに出力されるイベントは，ストレージに対する読み込み，または書き込みにより発生する．これらの操作により，ストレージはイベントの情報に基づいた状態遷移を行う．運用系の仮想マシンからストレージへのイベントを契機に同期をしなかった場合，待機系はこのイベント以前の状態から再開することになる．しかし，待機系は運用系のストレージに対する操作を知らないため，待機系とストレージの関係は整合性の無い状態になってしまう．従って，仮想マシンからストレージへのイベントを契機に運用系と待機系の同期を取らなければならない．

一方，ストレージから仮想マシンへのイベントは，仮想マシンがストレージを操作した結果として発生する．運用系の仮想マシンからストレージへのイベントを契機に同期をした場合，待機系はストレージを操作する直前の状態から再開する．待機系は，運用系がストレージに行った操作を再び実行するため，運用系が受け取ったものと同じ応答をストレージから受けることができる．従って，ストレージから仮想マシンへのイベントを契機に同期をする必要がない．

5. 実　　装

提案方式の一部を，オープンソースの仮想マシンモニタである Xen[7] に実装した．図 5 に，実装したプロトタイプの概観を示す．

5.1　Xen の概要

5.1.1　構　　成

Xen では，ゲスト OS を実行する環境をドメインと呼ぶ．ドメインには，他のドメインを管理する権限を持ったドメイン 0 と，そうではないドメイン U の 2 つの種類が存在する．Xen はハードウェア資源のうち，CPU とメモリの仮想化を行う．ドメイン 0 はドメイン U の操作に加え，I/O デバイスの仮想化も行う．このように，ドメイン 0 は特殊な権限を持つため，サービスを提供するサーバとして利用するのは適切ではない．従って，同期の対象はドメイン U のみとする．Xen が提供するドメインで OS を実行するためには，Para Virtualization と呼ばれる，OS の一部の命令は予め書き換え方式と，Full Virtualization と呼ばれる，プロセッサの仮想化支援機構を利用することで，書き換えを必要としない方式がある．ここでは，Para Virtualization を前提に説明する．

5.1.2　イベントチャネル

イベントチャネルとは，仮想化されたシステム内で通信を行う仕組みである．イベントチャネルを通じて，Xen とドメイン，または，ドメイン同士で通信することができる．イベントチャネルにはポートがあり，イベント毎に送受信するポートが決まっている．ドメイン U でやり取りされるイベントは，全てイベントチャネルを経由する．

5.1.3　I/O の仕組み

Xen によって仮想化されたシステムでは，特殊な権限を持つドメイン 0 のみが，I/O デバイスを直接操作できる．ドメイン U で実行されるゲスト OS は，デバ

イスを直接操作できないため，ドメイン0が提供する仮想デバイスを通じて処理を行う．仮想デバイスは，ドメインUが持つフロントエンドドライバと，ドメイン0が持つバックエンドドライバで構成される．ドメインUのフロントエンドドライバから送られてきたリクエストは，ドメイン0のバックエンドドライバによって，物理デバイスへのリクエストに変換される．フロントエンドドライバとバックエンドドライバ間の通信はイベントチャネルで実装されている．

5.1.4 マイグレーション

Xenはマイグレーションと呼ばれる，ドメインUを異なる物理マシンに移動する機能を持つ．特に，動作中のドメインUのメモリを転送し続け，ドメインUの停止時間を短くした機能は，ライブマイグレーション[9]と呼ばれる．

5.2 イベント捕捉機能

4節で提案した方式を実現するためには，ドメインUのイベントを捕捉する仕組みが必要になる．ドメインUは仮想デバイスのみを扱うため，フロントエンドドライバとバックエンドドライバを接続するイベントチャネルに着目し，ここを通るイベントを捕捉する機能を実装した．

5.3 仮想マシンの一時停止

運用系と待機系の同期は，運用系の動作を停止した上で行う．Xenには，ドメインUを一時停止する方法として以下の2つがある．

- suspend
- pause

suspendはマイグレーションで用いられている仕組みである．suspendは，ドメイン0の管理ツール，Xen，対象のドメインUが協調し，ドメインU自身で転送できる状態にする．しかし，suspendは非同期に行われるため，イベントを捕捉した瞬間にドメインUを停止させることができない．従って，検討方式を実現する方法として，suspendでは不十分である．

pauseは，ドメインUをXenのスケジューリングから外すことにより，ドメインUの動作を一時的に停止する仕組みである．Xenのスケジューラに対する操作だけで実現しているため，ドメインUを高速に停止することができる．しかし，現在のマイグレーションの仕組みにpauseを適用したところ，ドメインUを転送先で動かすことができなかった．機能としては未完全であるが，提案方式ではイベントの捕捉を契機にドメインUを停止する必要がある．従って，6節ではpauseを用いて評価を行う．

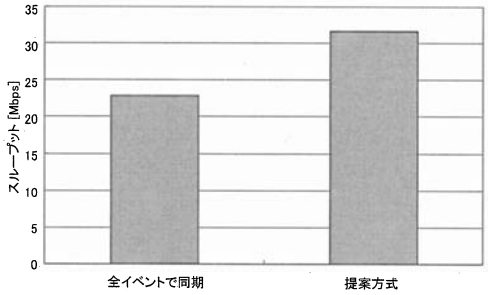

図6 ネットワークの性能

表4 同期処理の詳細（ネットワーク）

	回数	間隔 [ms]	ページ数	転送時間 [ms]	同期時間 [ms]
提案方式	8301	16.214	216	8.317	16.156
全イベント	16223	11.494	118	4.145	11.409

5.4 仮想マシンの継続的な差分転送

Xenのライブマイグレーションでは，まずメモリの差分を複数回に分けて転送し，続いてCPUのコンテキストに関する情報やドメインUがXenと共有するデータを転送している．一方で，提案方式ではメモリの差分の1回の転送と共に，CPUコンテキストやXenとの共有データの転送を必要とする．そこで，これらの情報を継続的に転送するよう実装した．

6. 評　　価

提案方式の効果を評価するため，3.2節で示した環境でネットワークとファイルI/Oの性能を測定した．また，同期処理に要する時間のに内訳ついても測定し，全イベントを契機に同期した場合と比較した．

6.1 ネットワークの性能

図6に，Netperfによる性能測定の結果を示す．提案方式を適用した場合，全てのイベントを契機に同期をした場合と比べ，スループットが39%向上した．提案方式の性能は，Xenで仮想化されたのみで同期を行わなかった場合の34%であり，目標としていた50%には達していない．

表4に，同期毎に行った処理の内訳を示す．ネットワークから仮想マシンに入力されるイベントのみを同期の契機とすることで，同期の回数を49%削減することができた．一方，同期の間隔に対する同期時間は全イベントを契機とした場合で99.3%，提案方式では99.6%となり，予想に反して若干増加した．また，1回の同期処理で転送したページの数が83%増加したのに対して，ページの転送にかかる時間は2倍に増加してしまった．

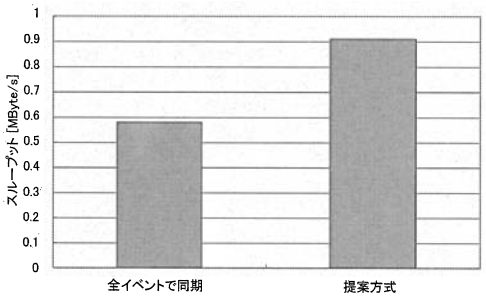

図 7　同期書き込みの性能

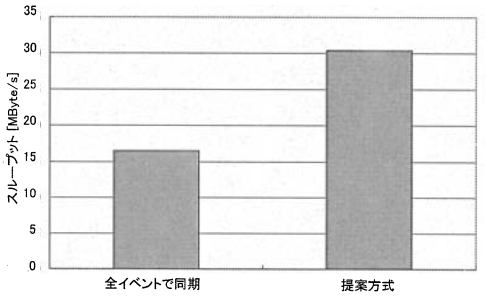

図 8　バッファリング付き書き込みの性能

表 5　同期処理の詳細 (同期書き込み)

	回数	間隔 [ms]	ページ数	転送時間 [ms]	同期時間 [ms]
提案方式	66020	11.652	81	2.508	9.461
全イベント	134356	9.607	45	1.125	7.947

表 6　同期処理の詳細 (バッファリング付き書き込み)

	回数	間隔 [ms]	ページ数	転送時間 [ms]	同期時間 [ms]
提案方式	92	391.007	3238	135.648	152.837
全イベント	3548	19.136	200	7.719	15.111

これらの結果から，同期の頻度を削減したことにより，同期中の仮想マシンにおけるネットワークの性能が改善されたことを確認できた．一方，同期間隔に対する同期時間が短くなっていないことが，十分な性能改善に結びつかなかった要因と考えられる．転送対象となるページ数の増加に比べ，転送時間の増加が大きかったことからも，1 回の転送処理における効率が落ちたと考えられる．

提案方式では，転送したページ数と転送時間から，運用系から待機系に約 812Mbps の速度でページが送信されている．運用系と待機系の接続には Gigabit Ethernet を使っており，帯域を十分に利用しきれていない．今回の実装では既存のライブマイグレーションを活用したが，提案方式に合わせたチューニングを行う必要がある．

6.2　ファイル I/O の性能

図 7 に，同期書き込みの測定結果を示す．提案方式の場合，全てのイベントを契機に同期をした場合と比べ，スループットが 57%向上した．提案方式の性能は同期を行わなかった場合の 16%であり，目標としていた 50%には達していない．

表 5 に同期毎に行った処理の内訳を示す．仮想マシンからストレージに入力されるイベントのみを同期の契機とすることで，同期の回数を 51%削減することができた．また，同期間隔に対する同期時間は全イベントを契機とした場合で 82.7%，提案方式では 81.2%となった．一方，1 回の同期処理で転送したページの数が 80%増加したのに対して，転送にかかる時間は 2.2 倍に増加した．

同期書き込みの場合，書き込み要求ごとにデータをストレージに書き出すため，多くのイベントが発生する．提案方式では，これに伴い待機系との同期が頻繁に行われるため．同期を行わなかった場合と比較して性能の低下が著しい．そこで，書き込み要求ごとにデータをストレージに書き出さない，バッファリング付き書き込みの性能を測定した (図 8)．提案方式の場合，全てのイベントを契機に同期をした場合と比べ，スループットが 84%向上した．提案方式の性能は同期を行わなかった場合の 56%であり，目標としていた 50%を達成している．

表 6 に同期毎に行った処理の内訳を示す．同期の回数は 97%削減され．同期の間隔に対する同期時間は全イベントを契機とした場合の 1/2 以下になった．一方，1 回の同期処理で転送したページの数が 16 倍に増加したのに対して，ページの転送にかかる時間は 18 倍に増加した．

これらの結果から，同期の頻度を削減したことにより，同期中の仮想マシンにおけるファイル I/O の性能が改善され，同期間隔に対する同期時間も短くなったことを確認できた．一方，ネットワークの場合と同様に，転送対象となるページ数の増加に比べ，転送時間の増加が大きかったことから，1 回の転送処理における効率が落ちたと考えられる．

同期書き込みの性能については，ネットワークと同様に転送処理のチューニングが必要である．一方，バッファリング付き書き込みでは転送対象となるページ数が多い．更なる性能改善のためには，ページの圧縮転送やイベント自体を待機系に転送する方法についても検討する必要がある．

7. 関連研究

8), 10), 12) では，仮想マシンモニタを利用し，アプリケーションや OS から見て透過的に可用性を向上する取り組みがなされている．8) では，仮想マシンの外部からのイベントや，仮想マシンで実行される命令のうち，演算結果が外部の状態に依存する命令を，運用系と待機系の仮想マシンモニタにシミュレートさせることで同期を行っている．しかし，仮想マシンモニタによるシミュレートの実装に PA-RISC[1] の recovery register を利用しているため，本研究が対象としている PC サーバには適用できない．11) は可用性の向上が目的ではないが，仮想マシンモニタによるシミュレーションを PC アーキテクチャに実装し，セキュリティの研究に応用している．10) では，仮想マシンの状態を継続的に外部ストレージに保存することを提案しているが，運用系がどの時点で停止しても待機系が同じ状態になる同期方式ではないため，待機系が処理を引き継ぐことができない場合がある．12) では，単一のシステムにある複数のプロセッサの同期を取ることで，一方のプロセッサが故障したときに，他方のプロセッサが処理を引き継ぐという方式を提案している．しかし，この方式ではプロセッサ以外の故障には対応できない．

4), 5), 6) は，仮想マシンによるサーバ統合環境で，高可用性を実現する商用製品である．これらにおける同期の契機はポリシーとしてユーザに委ねられており，アプリケーションや OS に依存した記述をする必要がある．

8. おわりに

本稿では，安価な PC サーバを対象に，仮想マシンによるサーバ統合環境で，アプリケーションや OS に依存しないで，障害発生時にサービスを継続する技術を提案した．さらに，提案技術に必要となる，低オーバヘッドな仮想マシンの同期方式を提案し，仮想マシンモニタである Xen を利用したプロトタイプを実装した．本研究では，仮想マシンを停止する方法として，既存の Xen の機能の利用を試みたが，提案方式を実現するためには，十分ではないことが明らかになった．今後，仮想マシンの停止方法について検討を行い，仮想マシンを外部から操作し，高速に転送状態にするための仕組みを実装する必要がある．また，転送処理の最適化や効率的な同期処理などを検討していく．

参考文献

1) HP PA-RISC architecture. http://www.hp.com/.
2) IOzone. http://www.iozone.org/.
3) Netperf. http://www.netperf.org/.
4) Virtual Iron. http://www.virtualiron.com/.
5) Vizioncore esxReplicator. http://www.vizioncore.com/.
6) VMware Infrastructure 3. http://www.vmware.com/.
7) Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 164–177, 2003.
8) Thomas C. Bressoud and Fred B. Schneider. Hypervisor-based fault tolerance. *ACM Transactions on Computer Systems*, Vol.4, No.1, pp. 80–107, February 1996.
9) Christopher Clark, Keir Fraser, Steven Hand, JacobGorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI '05)*, May 2005.
10) Brendan Cully and Andrew Warfield. Secondsite: Disaster protection for the common server. In *Proceedings of the Second Workshop on Hot Topics in System Dependability (HotDep'06)*, 2006.
11) George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza A. Basrai, and Peter M. Chen. Revirt: enabling intrusion analysis through virtual-machine logging and replay. In *Proceedings of the 5th symposium on Operating systems design and implementation (OSDI)*, pp. 211–224, October 2002.
12) Dominic Lucchetti, Steven K. Reinhardt, and PeterM. Chen. Extravirt: detecting and recovering from transient processor faults. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pp. 1–8. ACM Press, 2005.