2008－OS－107 (11)
2008／1／31

仮想機械のスナップショット機構を利用したサービスの高信頼なパッシブ複製手法

杉木 章義† Richard Potter† 加藤 和彦‡†

† 科学技術振興機構 CREST
‡ 筑波大学大学院 システム情報工学研究科
E-mail: {sugiki,potter}@osss.cs.tsukuba.ac.jp, kato@cs.tsukuba.ac.jp

要 旨

近年，PC サーバ上で仮想機械を利用したサービス提供が広く行われている．仮想機械はサービスの提供に必要な環境をカプセル化し，環境ごと移送できるため，ディザスタリカバリなどの障害対策にも応用することができる．本論文では，特定のサービスに依存しない仮想機械のスナップショット機構を利用した高信頼なパッシブ複製手法を提案する．本機構は Paxos アルゴリズムを利用し，スナップショットの複製中に障害が発生しても，最新のコミットされたスナップショットから正しく回復させることができる．また，本機構は差分スナップショットとそのマージ機構を利用しており，ネットワーク転送コストと回復時間の削減の両立を行っている．本機構を SBUML を利用し，Java で実装を行った．標準的なベンチマークで実験を行ったところ，障害から正しく回復できることを対処できることを確認した．

Reliable Passive Standby of Virtual Machines using a Generic Snapshot Mechanism

Akiyoshi Sugiki† Richard Potter† Kazuhiko Kato‡†

† CREST, Japan Science and Technology Agency
‡ Department of Computer Science, University of Tsukuba
E-mail: {sugiki,potter}@osss.cs.tsukuba.ac.jp, kato@cs.tsukuba.ac.jp

Abstract

The use of virtualization technology on commodity PCs has become popular recently. Strong encapsulation and migration functionalities of virtualization make it easy to achieve disaster-tolerant systems. This paper presents a cheap and service-independent state replication technique based on virtual machines. Our approach has two notable features. First, it is based on a well-known Paxos consensus algorithm and we use it for passive standby. Second, our mechanism reduces network transfer cost and recovery time by using incremental snapshots and their merge mechanism. We implmented our prototype in Java with SBUML virtual machines. Our expriments show that our mechanism can sufficiently recover from failures for two standard benchmark workloads.

1 はじめに

近年，PCサーバ上で仮想機械を利用してサービスを提供することが広く行われている．仮想機械は従来のハードウェアと互換性があり，同一サーバ上で複数の仮想機械を同時に稼働させることができるなど，サービス環境としての利用にはさまざまな利点がある．

一方で，仮想機械はディザスタ・リカバリなどの障害対策においても多くの利点がある．従来の障害対策はサービス・ソフトウェアを改変して実現していたのに対して，仮想機械の外部に複製などの障害対策機構を実現すれば，特定のサービスに依存しない高信頼化を実現することができる．また，仮想機械はライブラリやOSなどの実行環境の全てをカプセル化するため，サービス・ソフトウェアのみを高信頼化する場合に比べ，より安全である．さらに，仮想機械は実行中の動的な状態をそのままカプセル化するため，従来のストレージを利用した静的な回復手法に比べて短時間で回復できる可能性がある．

しかし，仮想機械を利用したサービス複製の実現には多くの解決しなければならない課題がある．まず，仮想計算機の状態を他の複数のサーバに複製する場合，複製の途中で障害が発生すると必ずしも最新の状態から回復するとは限らない．例えば，半数のサーバのみに複製が完了し，障害が発生した場合，どのサーバから回復するかによって結果が異なってしまう．また，全てのサーバに対する複製の完了を待ちながら処理を進めると，遅いサーバの性能や一時的な離脱に全体が支配されてしまう．さらに，障害発生時に1台の計算機のみが正しく代替することを保証しなければならない．

本論文では，仮想機械を利用したサービスの高信頼なパッシブ複製の手法を提案する．本手法は合意アルゴリズムの一種であるPaxos[1, 2]と仮想機械のスナップショット機構を結合し，上記の問題を解決する．本機構の特徴は次の2つである．

- **仮想機械のパッシブ待機**：本機構はスナップショット機構を利用し，プライマリ計算機の状態を他の計算機に複製するパッシブ待機を行う．Paxosを利用し，最新のコミットされたスナップショットから唯一の計算機が代替することを保証する．また，過半数の計算機から応答が得られればよいため，遅い計算機や一時的な障害の影響を避けることができる．パッシブ待機は，サービスが非決定性でもよい，計算コストが小さいなどの利点がある．
- **ネットワーク転送量と回復コストの削減**：本機構は仮想機械の差分スナップショット機構を利用し，状態の差分のみを転送する．この手法はネットワーク転送量を削減できる反面，多数の差分スナップショットが複製先に蓄積する．そのため，スナップショットのマージ機構を併用し，回復時間の削減を行う．

本論文の貢献は，次の3点にある．まず，仮想計算機やPaxos自身はよく知られているが，これらの機能をどう取捨選択し，組み合わせ，実際の耐障害性機構として構築するか明らかにした．また，その際に差分スナップショットの利用や段階的な回復試行など，可能な最適化や仮想機械ならではの利点について明らかにした．最後に，実際に性能の測定を行い，その可能性と限界について明らかにした．

本機構をJavaで実装し，評価を行った．仮想機械にはScrap Book for User-Mode Linux (SBUML)[3]を利用した．標準的なベンチマークで実験を行ったところ，障害に対して正しく回復することを確認した．

2 問題の分析

2.1 想定環境

本研究では図1のような環境を想定している．本機構をディザスタ・リカバリに利用することを想定しており，広域分散環境までを対象としている．この環境ではディスクの共有は行わず，それぞれのサーバが個別にディスクを持つ．よって，ディスクの共有を前提としているHAクラスタ[4, 5]とは想定が異なる．

また，本研究は安価な耐障害性の実現方法を目指すため，専用線ではない一般的なインターネット環境を想定する．この環境では回線の品質に対する保証がないことから，障害発生時にどの程度最新の状態から回復するかということと通常時のサービス処理性能の間にはトレードオフの関係がある．例えば，仮想機械内部のイベントをすべて捕捉し，複製の完了を待って処理を進めていたのでは，単一計算機で処理を行っていた場合に比べ性能が大幅に低下する可能性がある．よって，本研究では，常に最新の状

表 1: アクティブ待機とパッシブ待機の比較

手法	複製方式	サービスの種類	計算コスト	通信コスト	段階的な回復試行
アクティブ待機	リクエストを複製	決定性のみ	×	○	×
パッシブ待機	処理結果を複製	どちらでもよい	○	×	○

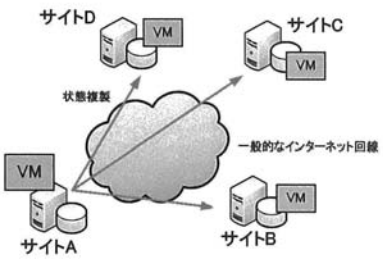

図 1: 広域環境でのサービスの状態複製

態から回復することよりも，これらのトレードオフの中でできる限り最新の状態から回復することを目指す．

2.2 アクティブ待機とパッシブ待機

従来から単一計算機で動作するサービスの耐障害性を高める手法としてサーバの**状態複製**（state replication）という手法が広く知られている．この手法はクライアントからは単一計算機で動作しているように見え，実際には複数の計算機に状態を複製する．この手法は従来の単一計算機で動作していたサービスを改変することなく，そのまま高信頼化できるという利点がある．

この手法には大きく分けて，アクティブ待機とパッシブ待機がある．それぞれの特徴を表 1 と次に示す．なお，これらの派生としてセミ・アクティブ待機，セミ・パッシブ待機などがあるが，基本的にいずれかの手法を元にしている．

- **アクティブ待機**：アクティブ待機ではリクエストを複製し，それぞれの計算機で独立にリクエスト処理を行う．この手法は個別にリクエスト処理を行うため計算コストは高いが，通信コストは低いという利点がある．また，それぞれの計算機が同一の入力から同じ状態へと遷移することを仮定しているため，サービスが決定的である必要がある．
- **パッシブ待機**：パッシブ待機では，**プライマリ計算機**でリクエストを処理し，その実行後の状態を**バックアップ計算機**に複製する．プライマリ計算機の処理結果を元に状態遷移が行われるため，サービスは非決定性でもよく，バックアップ計算機の計算コストも小さい．その反面，要求に比べリクエスト処理結果の方が一般に大きいため，通信コストが大きいことが知られている．

本論文ではパッシブ待機の手法を用いる．本手法は汎用的なサービスを想定しているため，サービスの決定性を仮定するのは難しい．また，バックアップ計算機の計算コストが小さいため，サービスが複数あった場合にそれらのバックアップ計算機をまとめ，少数の計算機で運用することができる．さらに，最新のスナップショットから回復できなかった場合，過去のスナップショットをたどり，順に回復を試行することができる．

近年，仮想機械でサーバの状態複製を行う研究が行われ始めているが，これらは本手法とは異なる．VM-FIT[6] はアクティブ待機を行っており，[7] は Xen でセミ・アクティブ待機を行っている．Second Site[8] は Xen でイベント単位での複製を行い，パッシブ待機に近いが，ポジションペーパやスライドのみが示されており，その詳細は明らかではない．特に，複数の計算機間でどのように最新のスナップショットからの回復を保証しているか，そのプロトコルは不明である．

3 仮想機械のパッシブ待機

本論文では仮想機械のスナップショット機構を利用した高信頼なパッシブ待機手法を提案する．本機構は合意アルゴリズムの一種である Paxos[1, 2] を使用し，スナップショット転送中であっても最新のコミットされたスナップショットから正しく回復させることができる．

本機構では，Paxos による複製機構をホスト OS

上で動作させる．また専用のゲストOSを用意し，その上で動作させることも可能である．Paxosでは何らかの**障害検出器**によりサービスの障害が検出できるものとし，ホストOSはFail-Stop障害のみを仮定する．これは最も基本のPaxosを使用しているためであり，さまざまに提案されている改良[9]によりこれらの障害以外にも対処することができる．一方，仮想機械内のゲストOSは障害検出器によって検出できれば，Fail-Stop障害に限らず対処することができる．現状ではF台の障害に対処するために$2F+1$台の計算機が必要であるが，[10]などの改良により緩和することができる．なお，今回Paxosは[11]の資料を基に作成した．本手法はPaxosのプロトコル自身は改変しないため，オリジナルのプロトコルが持つsafetyとlivenessは保持している．

3.1 概要

Paxosでは，**ビュー**と呼ばれる複数の計算機からなるグループに対して同一の操作を行う．ビューはクライアントからは単一の計算機のように見える．障害が発生した場合，ビューに参加している計算機の変更が行われる．ビューには論理時計による番号がついており，ビューに変更があった場合には論理時計が進められ，以前のビューの処理は以後受け付けられない．本機構では論理時計にLamport時計[12]を利用している．

Paxosの処理は2つの状態から構成される．(a)ビュー上で通常の複製処理を行う**通常モード**と(b)障害からの回復処理を行う**ビュー変更モード**である．以下，それぞれの処理を順に説明する．

3.2 通常時の動作

通常時のスナップショット複製処理の流れを図2に示す．プライマリ計算機の仮想機械上でサービスが稼働しており，複数のバックアップ計算機にその状態をスナップショットとして複製する．

まず，(1)プライマリ計算機で差分スナップショットの取得を行う．(2)プライマリ計算機は直ちにスナップショットをログに記録し，複数のバックアップ計算機に転送する．(3)バックアップ計算機はスナップショットをログに記録すると，プライマリ計算機に応答を返信する．(4)プライマリ計算機は過半数以上のバックアップ計算機から応答を受け取ると，

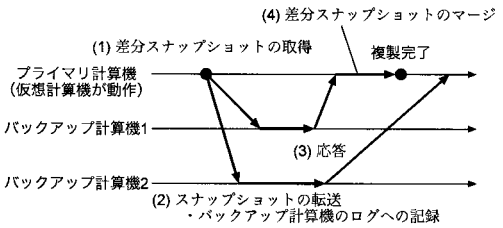

図2: 通常時のPaxosの動作

コミット処理として差分スナップショットをマージし，完全スナップショットを作成する．バックアップ計算機では，次回スナップショット複製時の(2)の段階で，プライマリ計算機がマージしたところまでのスナップショットをコミットし，完全スナップショットを作成する．

このように，Paxosでは前回までのスナップショット処理と重複させながら非同期的に処理が進められていく．また，要求には番号がついており，前回までのすべてのスナップショットを受け取るまで以降のマージ処理が待たされるため，スナップショットの整合性が保たれる．さらに，過半数の計算機から応答があればよいので，遅いノードの影響を避けることができる．遅れている計算機は，次節で説明するビューの変更時に追いつく．

この手法は，差分スナップショットを用いるので完全スナップショットを転送する場合に比べてネットワークの転送量が削減されている．また，差分スナップショットが大量にバックアップ計算機に蓄積すると，回復処理や通常のサービス処理性能に大きな影響を与えるが，本機構はコミット可能となった時点でマージを行うため，これらのコストを抑えることができる．

3.3 障害からの回復時の動作

障害回復処理では，新しいプライマリを選出し，サービスを継続する．障害回復は図3のように，2往復のメッセージ送受信で行う．最初に(a)次回のビューを決める計算機を決定し，次に(b)その計算機が新しいビューを提案し，ビューの変更を行う．

最初のステップでは，(1)仮想機械の障害を検出したいずれかの計算機がビューの変更を要求する．このとき，複数の計算機が同時に要求を行ってもよい．この要求は次回のビュー番号の候補を含んでお

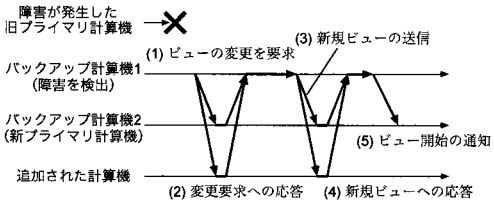

図 3: 障害回復時の Paxos の動作

り，(2) 受信側は最も高いビュー番号を送信したものに対して応答を返し，この番号より低い計算機に対して応答しない．(3) 過半数以上から応答を受け取った障害検出ノードは次のステップに進み，新しいビューを送信する．ここでも同様に (4) 過半数から応答が得られれば，新しいビューを確立し，(5) 新しいプライマリ計算機に新しいビューの開始を通知し，通常状態に移行する．

このとき，最新の状態を持っている唯一の計算機がプライマリ計算機として選出される．また，処理が遅れていた計算機は，(4) の前に (3.5) として他の計算機からこれまでの処理結果を完全スナップショットとしてもらってから新しいビューに対する応答を返すため，最新の状態に追いつくことができる．一般の Paxos における処理では，これまでの要求のすべてを転送する必要があるが，仮想機械では最新のコミットされた完全スナップショットを転送すれば，追いつくことができるため，簡単である．新しくプライマリ計算機に選ばれた計算機は完全スナップショットから仮想機械を起動し，定期的な差分スナップショットの複製を行う．

3.4 段階的な回復試行

パッシブ・スタンバイのもう一つの利点として，最新のスナップショットからサービスを回復できなかった場合，過去のスナップショットを段階的にさかのぼり回復を試行することができる．この手法により，タイミングなどのバグから発生する非決定性の障害を乗り越えることができる．一時的障害を乗り越える点で，Rx[13] や ExtraVirt[14] などに近い．

本機構では，以下のようにしてこの機能を実装することができる．差分スナップショットのマージ処理を完了した後，直ちにログを削除せず，ある一定期間，過去のスナップショットを保存しておく．障害回復時に最新のスナップショットから起動できなかった場合，Paxos を通じて 1 つ前のスナップショットに戻すことを他の計算機に要求しながら，試行する．この試行段階でさらなる障害が発生しても，複数の計算機にどの時点のスナップショットまで戻したか保存されているため，その続きから再試行を継続することができる．

4 実装

本機構を本研究室で開発しているサステーナブルサービス基盤 [15] の一部として実装した．生産性を優先するため Java で実装しており，10,292 行で構成されている．

本機構は Java で実装されているが，高速化のためにさまざまな工夫を行っている．本機構は内部を処理ごとにスレッドに分割し，イベント駆動で処理を行う．ちょうど SEDA[16] のような構成となっている．また，通常のメッセージ送受信には Java のシリアライズを利用するが，仮想機械のスナップショットなど大きなファイルの転送には sendfile システムコールに対応する I/O 機能で転送する．これらの送受信を混在させても，メッセージの FIFO 順序は保たれる．さらに，TCP 接続は接続されたままプールされており，単一スレッドで複数の TCP 接続の処理を行う．

仮想機械には SBUML を利用した．SBUML は User-Mode Linux の派生版であり，オリジナルがサポートしていないスナップショット機能を提供している．SBUML は TUN/TAP デバイス [17] をサポートする Linux であればどの環境でも利用可能である．TUN/TAP デバイスは標準的な Linux ディストリビューションであれば組み込まれた状態で配布されており，多くの環境で利用できる．なお，本機構は SBUML 以外の仮想機械も利用可能であり，現在 Xen[18] への対応作業を行っている．

5 実験

実験では，Paxos による複製機構のオーバヘッド，SBUML のオーバヘッド，全体を統合した場合の性能の 3 つの観点から行った．本研究は広域分散環境までも想定しているが，今回は複製機構自身や仮想機械によるオーバヘッドを調査するため，LAN 環境で評価を行った．

5.1 実験環境

実験には5台のCPU Quad Xeon X3210 2.13GHz, メモリ2GB, ディスク160GB (シリアルATA接続, 7200 rpm) で構成された計算機を使用した. 全ての計算機は1000BASE-Tで単一のスイッチに接続されている. ホストOSはFedora 8 (Linux 2.6.23) を使用し, JavaはSun JDK 1.6.0を使用した. ゲストOSはCentOS 3.9 (Linux 2.4.24-1um-1sb) を使用し, 実験中は全て256MBのメモリを割り当てた.

実験のベンチマークはdbench 3.0.4[19]とRUBiS 1.4.3[20]を使用した. dbenchはsambaの負荷を模倣するベンチマークである. 書き込みが90%に近い割合を占めるため, SBUMLにとって非常に負荷の高いベンチマークである. 一方, RUBiSはeBayオークションを模倣したベンチマークであり, 広く評価に利用されている. RUBiSは書き込みの比率が15%程度である. dbenchはクライアント数を4, RUBiSはクライアント数を50とした.

5.2 複製機構の性能測定

まず, 仮想機械によらない複製機構自身のオーバヘッドを調査するため実験を行った.

本実装のPaxosで, 何も処理を行わないnullリクエストを3台に複製した場合の所用時間を示す. 実験を行ったところ, 10,000回の平均は2.78[ms]であった. 文献[21]でC言語で記述した場合の単純なキーと値の複製で平均2.98[ms]程度であることが報告されており, Javaで実装したことによるオーバヘッドは小さいと言える. また, ビューの変更にかかる時間は平均69.79[ms]であった.

5.3 仮想機械の性能測定

次に仮想機械SBUMLによるオーバヘッドを示す. スナップショット取得間隔を30, 60, 120秒と変えながら, 平均スナップショット・サイズ, 平均スナップショット取得時間, サービスの平均可用性の3つを10分間にわたり測定した.

表2はdbenchワークロードの結果である. まず, 平均スナップショット・サイズを見ると差分スナップショットは完全スナップショットの6割程度と小さい. 一方, スナップショットの取得時間は処理コストの増加のため, 差分スナップショットの方が時間を要している. SBUMLではスナップショットを取得している間, 仮想機械を凍結させるため, サービスが一時停止する. スナップショット取得間隔に占めるスナップショット取得時間からサービスの利用可能時間を求めると, 表の可用性のような値となる. スナップショット取得間隔を大きくするにつれて可用性は向上していくが, 一方で障害が発生した場合に, より以前のサービス状態へと戻りやすくなる.

表2: dbenchでのスナップショット取得結果

	スナップショット取得間隔 [s]		
	30	60	120
サイズ [MB]	101.5	100.1	101.4
	59.6	59.8	60.0
取得時間 [s]	3.22	2.55	3.53
	4.32	4.14	4.24
可用性 [%]	89.3	95.7	97.1
	85.6	93.1	96.5

上段：完全, 下段：差分スナップショット

また, 差分スナップショットのマージによる効果について示す. dbenchワークロードで差分スナップショットを30秒おきに20回取得した場合, 差分スナップショットの合計は1,151MBに達するが, マージの効果により106MBの大きさの完全スナップショットとすることができる.

5.4 サービス複製の性能測定

最後に, Paxosと仮想計算機を結合したサービス複製の性能測定結果を示す. RUBiSベンチマークの負荷に対して3台の計算機で実験を行っている.

図4にプライマリ計算機に障害を発生させた場合の結果を示す. 180秒間実験を行っており, 40秒のあたりで障害を発生させている. 仮想計算機自身は5秒程度で代替処理を行っているが, RUBiSのクライアントがサーバに再び接続するまでの時間が長く, 実際にスループットが回復するまで30秒程度要している. これは本方式の問題というよりも, RUBiSのクライアントとのネットワークの問題によるところが大きい. また, 障害が発生していない状態であってもスループットが30秒程度おきに低下している地点があるが, これはスナップショット

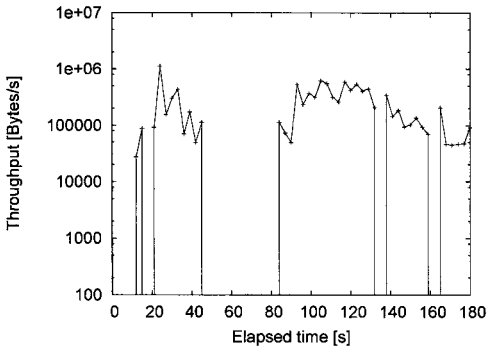

図 4: プライマリ計算機の障害からの回復結果

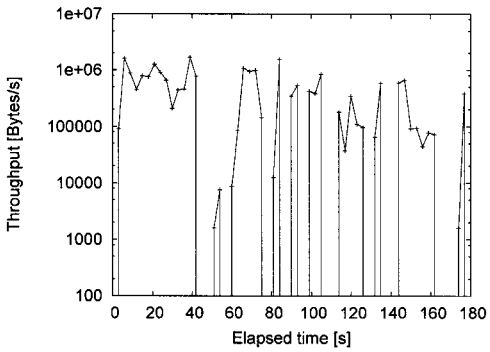

図 5: バックアップ計算機の障害からの回復結果

の取得を行っている地点である．

図 5 にバックアップ計算機に障害を発生させた場合の結果を示す．すぐに代替しなければいけないプライマリ計算機とは異なり，バックアップ計算機の障害は 1 台であれば残りの 2 台の計算機でビューを変更することなく代替することができる．

6 関連研究

広域分散環境を想定し，仮想機械を利用したディザスタ・リカバリ機構として Second Site[8] がある．Second Site はポジションペーパーのみが公開されており，実装の詳細が明らかにはなっていない．最近，公開されたスライド [22] によると Xen の live migration 機能 [23] を改変し，仮想機械内のメモリ操作やディスク操作などイベント単位での複製を行っていることが示されている．しかしながら，バックアップ計算機を複数とした場合に，どのように最新の状態からの回復を保証するか，そのプロトコルは示されていない．

Stodden ら [7] は Xen でセミ・アクティブ複製を行っている．通常のアクティブ複製では，割り込みや I/O などのイベントの処理順序による影響を受けるため，プライマリ計算機とバックアップ計算機でイベントの処理順序を同期させている．VM-FIT[6] では，Xen でアクティブ複製を行っている．これらの研究はアクティブ待機を元にしており，パッシブ待機を行う本研究とは異なっている．

VMware HA[4] や田村ら [5]，文献 [24] は LAN 内でディスクを共有する HA クラスタについて扱っている．本研究はディスクを共有しない広域分散環境を目指しており，これらの研究とは目的が異なる．

仮想機械によらないアクティブ待機・パッシブ待機の研究は従来から行われており，目新しい分野ではない．しかしながら，仮想機械を用いることでサービスを環境ごとカプセル化し，特定のサービスに限らない複製手法を実現することができるなど，多くの可能性がある．

7 議論と今後の予定

本機構は複数サービスに対応しており，複数の仮想計算機の状態を異なるサイトに同時に並行して複製することができる．この機能を利用し，複数拠点がお互いに資源を提供し合うことで，少ない計算機資源で高信頼化することができる．現在，このための作業を行っており，評価を進めている．

本機構は一般的なスナップショット機能を利用しているため，copy-on-write などの工夫により，仮想機械のスナップショット機能が高速化されれば，性能を大幅に改善させることができる．スナップショット機能の高速化は本研究に限らず，セキュリティなどのさまざまなアプリケーションで活用できるため今後の研究の進展が期待される．

本手法の回復処理は最新のスナップショットから行われるため，最後のスナップショットから障害発生までに到着したリクエストはわずかながら失われる可能性がある．しかしながら，このリクエスト損失は本手法とアクティブ待機と組み合わせることで防ぐことができる．クライアントと仮想計算機上のサーバの間にプロキシを設置し，最後のスナップショットからのリクエストをログに保存しておき，障害回復時にリクエスト処理を再度行うことで回復

させることができる．この手法を広域環境で実現する方法は今後の課題である．

8 まとめ

本論文では仮想機械のスナップショット機構を利用したサービスの高信頼なパッシブ複製手法を提案した．本機構はPaxosアルゴリズムを利用し，プライマリ計算機からのスナップショットの転送中に障害が発生しても最新のコミットされた状態から正しく回復させることができる．また，本機構は差分スナップショットとその結合機構を利用し，ネットワーク転送量と回復時間の削減を行っている．さらに，本手法は一般的な仮想機械のスナップショットを利用しているため，スナップショット機構を提供している仮想機械であれば広く利用することができる．

本機構のプロトタイプをJavaで実装し，評価を行った．仮想計算機にはSBUMLを利用している．さまざまな標準ベンチマークで実験を行ったところ，障害から正しく回復することが確認された．

謝辞

本研究は科学技術振興機構CREST「自律連合型基盤システムの構築」の支援を受けている．

参考文献

[1] Lamport, L.: The Part-time Parliament, *ACM TOCS*, Vol. 16, No. 2, pp. 133–169 (1998).

[2] Lamport, L.: Paxos Made Simple, *ACM SIGACT News*, Vol. 32, No. 4, pp. 51–58 (2001).

[3] Potter, R.: Scrap Book for User-Mode Linux. `http://sbuml.sourceforge.net/`.

[4] VMware Inc.: VMware High Availability. `http://www.vmware.com/products/vi/vc/ha.html`.

[5] 田村芳明，佐藤孝治，木原誠司，　盛合敏：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，情報処理学会研究会報告 (2007-OS-105)，pp. 71–78 (2007).

[6] Reiser, H. P. and Kapitza, R.: VM-FIT: Supporting Intrusion Tolerance with Virtualisation Technology, *1st Workshop on Recent Advances on Intrusion-Tolerant Systems* (2007).

[7] Stodden, D.: Semi-active Workload Replication and Live Migration with Paravirtual Machines, *Xen Summit, Spring 2007* (2007).

[8] Cully, B. and Warfield, A.: SecondSite: Disaster Protection for the Common Server, *USENIX HotDep 2006* (2006).

[9] Castro, M. and Liskov, B.: Practical Byzantine fault tolerance, *USENIX OSDI'99*, pp. 173–186 (1999).

[10] Lamport, L. and Massa, M.: Cheap Paxos, *IEEE/IFIP DSN 2004*, Los Alamitos, CA, USA, IEEE Computer Society, pp. 307–314 (2004).

[11] Mazières, D.: Paxos Made Practical (2007). `http://www.scs.stanford.edu/07wi-cs244b/sched/readings/paxos.pdf`.

[12] Lamport, L.: Time, clocks, and the ordering of events in a distributed system, *Comm. of ACM*, Vol. 21, No. 7, pp. 558–565 (1978).

[13] Qin, F., Tucek, J., Zhou, Y. and Sundaresan, J.: Rx: Treating bugs as allergies — Safe method to survive software failures, *ACM TOCS*, Vol. 25, No. 3, p. 7 (2007).

[14] Lucchetti, D., Reinhardt, S. K. and Chen, P. M.: ExtraVirt: Detecting and recovering from transient processor faults, *ACM SOSP'05 WiP* (2005).

[15] 小磯知之，阿部洋丈，鈴木与範，Potter, R.，　池嶋俊，加藤和彦：サステーナブルサービスのための基盤ツールキットの設計，情報処理学会論文誌：コンピューティングシステム，Vol. 48, No. SIG3 (ACS 17), pp. 13–26 (2007).

[16] Welsh, M., Culler, D. and Brewer, E.: SEDA: an architecture for well-conditioned, scalable internet services, *ACM SOSP 2001*, pp. 230–243 (2001).

[17] Krasnyansky, M. and Yevmenkin, M.: Universal TUN/TAP Driver. `http://vtun.sourceforge.net/tun/`.

[18] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualization, *ACM SOSP 2003*, pp. 164–177 (2003).

[19] Tridgell, A.: dbench. `http://samba.org/ftp/triage/dbench/`.

[20] ObjectWeb: RUBiS Benchmark. `http://rubis.objectweb.org/`.

[21] Lorch, J. R., Adya, A., Bolosky, W. J., Chaiken, R., Douceur, J. R. and Howell, J.: The SMART way to migrate replicated stateful services, *EuroSys '06*, New York, NY, USA, ACM, pp. 103–115 (2006).

[22] Cully, B.: High-speed Checkpointing for High Availability, *Xen Summit 5* (2007).

[23] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I. and Warfield, A.: Live Migration of Virtual Machines, *ACM/USENIX NSDI 2005*, pp. 273–286 (2005).

[24] Bressoud, T. C. and Schneider, F. B.: Hypervisor-based fault tolerance, *ACM TOCS*, Vol. 14, No. 1, pp. 80–107 (1996).