Quasar: CPU エミュレータ QEMU を利用した移動計算システム

Quasar: A Mobile Computing System Based on CPU Emulator QEMU

尾上 浩一 [1]　大山 恵弘 [1,2]　米澤 明憲 [1,2]
Koichi Onoue [1]　Yoshihiro Oyama [1,2]　Akinori Yonezawa [1,2]
[1] 東京大学 大学院 情報理工学系研究科
[2] 科学技術振興機構 戦略的創造事業
[1] Graduate School of Information Science and Technology, University of Tokyo
[2] CREST, Japan Science and Technology Agency
{koichi,oyama,yonezawa}@yl.is.s.u-tokyo.ac.jp

概 要

CPU エミュレータ QEMU を利用し，移動計算環境を提供するシステム Quasar を実装した．Quasar を用いると，QEMU が作る仮想計算機を別の物理計算機へ簡単にかつ効率的に移動させることができる．Quasar の特徴は仮想計算機上でのサーバホスティングを支援する機能群である．その 1 つは仮想ネットワーク機能であり，クライアントに意識させずにサーバを移動させることを可能にする．もう 1 つは仮想計算機の実行と転送を並列に行う機能であり，サーバの移動に伴うサービス停止時間を短縮する．我々は Quasar を Linux 上に実装し，実験を行った．実験では，サービス停止時間を非常に短く抑えてサーバを移動させることができることを確認した．

1　はじめに

近年，仮想計算機の技術が発展し，仮想計算機の性能と機能が著しく向上した．従来，仮想計算機は大型計算機の分野を中心に利用されてきたが，最近では PC などの計算能力が比較的低い計算機上でも，実用的な実行速度で仮想計算機を利用することが可能になってきている．仮想計算機には，独自の命令セットを提供するもの（Java [5], .NET [6] など），物理計算機のハードウェアを仮想化・多重化して提供するもの（VMware [8] など），実在のハードウェアをソフトウェアによって模擬するいわゆる CPU エミュレータ（bochs [7], QEMU [4] など）がある．

CPU エミュレータは，単純な実装ではオーバヘッドが大きいという欠点を持つものの，物理計算機への依存度が小さい仮想的な実行環境を作れるという利点を持つ．この利点は，様々な異なる物理計算機の上に共通の計算環境を構築したい場合や，計算環境を物理計算機間で移動させたい場合に特に重要である．

本研究では CPU エミュレータ QEMU を利用して実装した移動計算システム Quasar について述べる．Quasar は，QEMU が作る仮想計算機を，実行状態を保持したままで別の物理計算機に移動させることを可能にする．その際には多様な CPU をもつ物理計算機上で利用可能であるが，現在は OS が Linux であることが必要である．今後 Quasar を発展させ，たとえば，1 つの仮想計算機を Linux/IA-32 マシン，Mac OS X/PowerPC マシン，Solaris/SPARC マシンの間で移動させながら利用することを目指している．Quasar には仮想ネットワークの機能が備えられている．仮想計算機上で動作するソフトウェアとその通信相手のソフトウェアは，仮想計算機が物理的に移動しても，その移動を意識することなく通信を継続することができる．

Quasar の応用のひとつとしては，移動可能なサーバの運用が挙げられる．サーバの移動は，通信遅延の削減や物理計算機の停止・障害の隠蔽に有用である．Quasar には，仮想計算機上で運用されているサーバの移動に伴うサービスの停止時間を短縮するための処

理が含まれている．また，他の応用としては，職場のPC 上で使用していた仮想計算機を自宅の PC に転送し，自宅で同じ仮想計算機を利用して作業を継続することなどが挙げられる．

CPU エミュレータを移動計算システムの基盤として用いる利点を以下に述べる．第一には，移動するプログラムを様々な物理計算機上で実行できることである．VMware を利用した場合には物理計算機の CPU が IA-32 であることが要求される．QEMU を利用した場合には，IA-32 や PowerPC などの物理計算機上で仮想計算環境が利用可能であり，物理計算機上の CPU に対する依存度が小さい．第二には，普段物理計算機上で直接実行している既存のソフトウェアをそのまま利用できることである．Apache や Firefox などのアプリケーションおよび Windows や Linux などの OS を修正することなしに QEMU 上で動かすことができる．Java や. NET のような独自の命令セットを持つ仮想計算機と異なり，QEMU は IA-32 や PowerPC などの実在の CPU を模擬することにより，これを可能としている．また，Java や.NET 上で Linux などの OS を動作させることは現在のところ難しい．その結果，Java や.NET はアプリケーション単位での ‘run anywhere’ を提供するが，QEMU および Quasar は，OS 全体の計算機を単位とする ‘run anywhere’ を提供できる．

これまで様々な CPU エミュレータが提案されているが，我々の知る限り，移動計算システムへ応用したものはない．本研究では CPU エミュレータを移動計算システムに適用し，その有用性を評価するという一面もある．

本論文の構成は以下の通りである．2 章で CPU エミュレータ QEMU の概要について述べる．3 章で移動計算システム Quasar について述べる．4 章で実験結果を示す．5 章で関連研究について説明し，6 章でまとめと今後の課題について述べる．

2 CPU エミュレータ QEMU

2.1 概要

QEMU は Full system emulation, User mode emulation という 2 つの操作モードを提供している．Full system emulation を利用した場合には，CPU を含む計算機ハードウェア全体が仮想的に提供される．これにより，利用者は様々な CPU や OS を利用することができる．User mode emulation を利用した場合には，ある特定の OS のバイナリを異なる CPU 上で利用することができる．本研究の目的は，物理計算機に対する依存度が小さい移動計算システムを構築することであるため，Full system emulation を利用した．

以降では，物理計算機の計算環境をホスト環境と呼び，仮想計算機が提供する計算環境をゲスト環境と呼ぶ．また，ホスト環境上で走る OS をホスト OS と呼び，ゲスト環境上で走る OS をゲスト OS と呼ぶ．QEMU が実装されているホスト環境には，IA-32, PowerPC, Alpha, SPARC, ARM などがある．QEMU が Full system emulation を提供できるゲスト環境には IA-32, PowerPC などがある．

CPU エミュレータはハードウェア全体を模擬するため，実行性能が低下する傾向があるが，QEMU はこれを改善するため，いくつかの最適化を行っている．そのひとつとして動的バイナリ変換を利用している．QEMU は，ゲスト OS やアプリケーションの命令列をホスト OS およびホスト環境用の命令列に動的に変換する．変換された命令列はホスト環境のハードウェア上で直接実行される．

2.2 実行状態の保存・復元機能

QEMU には，メモリやデバイスの状態を含む仮想計算機の実行状態を保存・復元する機能がある．実行状態をファイルの形で保存することができ，そのファイルを利用して実行状態を，保存を行った同一物理計算機上または，異なる物理計算機上で復元することが可能となる．Quasar はこの保存・復元処理を利用して実装されている．

QEMU で実行状態の保存・復元のためには 2 つのファイルを利用する．ひとつは仮想計算機のファイルシステムの情報を保存する仮想ディスクファイルである．もうひとつは CPU，メモリ，キーボード，割り込みコントローラ，ネットワークカード（NIC）などの状態を保持する実行時コンテキスト情報ファイルである．

2.3 移動計算システムへの適用と問題点

仮想計算機を移動させるための方法を以下に述べる．まず，仮想計算機の実行時コンテキスト情報をファイルに保存する．次に，仮想ディスクファイルとともに別の計算機に転送するか，または 2 つのファイルを CD-RW や DVD などの外部記憶装置に保存し，別の計算機に移動する．そして，移動先の計算機上で仮想環境を復元し，仮想計算機の移動が完了する．

この処理自体は QEMU を修正することなく実現可能である．ただし，移動計算システムの基盤として用いるには，いくつかの機能の不足や改善すべき点があり，QEMU を拡張する必要がある．特に，QEMU が提供する仮想計算機の上に，移動可能なサーバを構築しようとした場合に不便な点に直面する．まず，QEMU はそもそも移動計算を目的として開発されたシステムではないので，仮想計算機の実行状態の保存・転送・復元を自動的に順番に実行するための仕組みが存在しない．また，QEMU はネットワークを仮想化する機能を提供していないため，仮想計算機が移動する際には接続が切断されるとともに，ゲスト OS に関係づけられる IP アドレスが変わる．その結果，移動を意識していないアプリケーションは正常に動作しなくなることがある．さらに，仮想計算機で利用するメモリなどの実行状態を表現するデータサイズや転送に必要な仮想ディスクのデータサイズが大きい場合，保存・転送・復元の処理に時間がかかる．その結果，サーバを移動させる際にはサービスを長い時間停止せざるをえなくなる．

3 移動計算システム Quasar

3.1 概要

Quasar は以下の機能を提供する．

- 仮想計算機を移動させるための処理，すなわち，仮想計算機の実行状態の保存・転送・復元に関する一連の処理を自動的に行う機能

- 仮想ネットワーク機能．ホスト OS に割り当てられている IP アドレスとは異なる IP アドレスをゲスト OS に割り当てることができる．ホスト OS の IP アドレスとゲスト OS の IP アドレスは独立している．よって，ゲスト OS は，物理計算機間を移動しても，同じ IP アドレスのままで外部の計算機と通信を続けることができる．また，移動の際に通信が切断されることなく，接続を維持することも可能である．

- 仮想計算機の高速な移動．仮想計算機上でサーバを運用する用途において，物理計算機間の移動に伴う，仮想計算機上で動作しているサービスの停止時間を削減する機能．仮想計算機の実行状態の転送を二段階に分け，第一段階ではサービスの提供と転送処理を並列に行うことによりこれを実現する．

以下で Quasar の実装の詳細について説明する．

3.2 実行状態の保存・転送・復元機能

Quasar では，ユーザは移動元のホスト OS の IP アドレスを指定し，移動要求を出すことにより簡単に仮想計算機を移動することができる．Quasar は移動要求を受け取ると，移動元の物理計算機上で現在実行中の仮想計算機の実行状態を保存・転送し，移動先の物理計算機上で実行状態を復元する．これらは連続的にかつ自動的に実行される．

我々は，アプリケーションおよびゲスト OS のエミュレーション実行と，独自ネットワーク処理，仮想ネットワーク処理，移動処理を並列に実行させるために，各処理操作を行うスレッドを導入した．移動元，移動先の物理計算機上では，それぞれ移動デーモン，移動クライアントが存在する．移動デーモンは移動要求を受け取り，仮想計算機の実行状態の保存・転送を行う．移動クライアントは移動要求の発行，保存データを受け取り，実行状態の復元を行う．

我々は，異なる物理計算機への移動の前後において，実行時コンテキスト情報のうち保存・復元しない方がよいものを取り除いた．現在の実装では，Real Time Clock (RTC) と CPU のクロック数情報であり，これらは移動後の物理計算機の情報を利用した方がよい．RTC は再開したときの時間を利用することで現在時刻を正確に設定することができる．また，移動後の物理計算機のクロック数情報を利用することにより，ゲスト OS のタイマー割り込みなどがより正確に行われる．これを実現するため，QEMU の保存・復元操作を

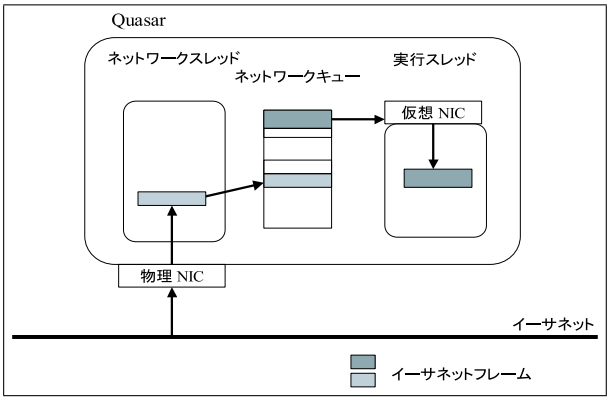

図 1: イーサネットフレームの受信処理

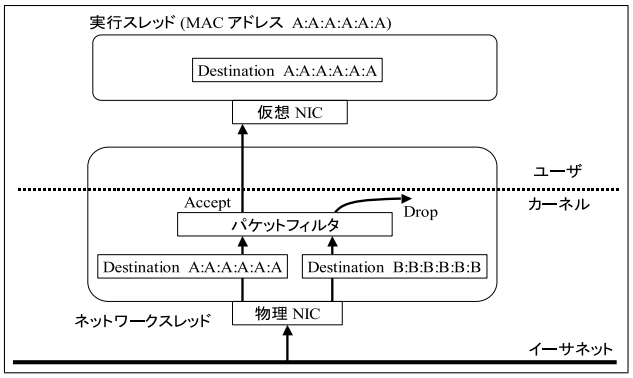

図 2: カーネルレベルでのパケットフィルタリング処理

改良した．

Quasar の基本的な利用形態では，仮想計算機を動作させる可能性がある物理計算機上には，初期データを含んだ仮想計算機の仮想ディスクをあらかじめ準備しておく．移動の際には，ゲスト OS 起動時から変更のあった仮想ディスクのセクタのみを転送する．

3.3 仮想ネットワーク

Quasar は仮想ネットワーク機能を含んだ独自のネットワーク機能を提供している．

ユーザは Quasar を利用するときにゲスト OS の仮想 NIC に対する仮想 MAC アドレスを指定するだけでよい．これにより，DHCP サーバから得られる動的なグローバル IP アドレスを利用することも可能となる．仮想ネットワークを含めたネットワーク機能を利用するために，ホスト計算環境を変更する必要がなく，ゲスト OS 内でグローバル IP アドレスなどのネットワークに関する設定をするだけでよい．

Quasar の中では，実装上，仮想計算機の実行を行うスレッド（実行スレッド）とともに，イーサネットフレームの処理を行うスレッド (ネットワークスレッド)，イーサネットフレームを移動先へ転送する処理を行うスレッド（仮想ネットワークスレッド）が走っている．

独自のネットワーク機能はデータリンク層で実現している．Quasar は物理 NIC を promiscuous mode で走らせることにより，ネットワーク上を流れるすべてのデータを Quasar が受信することができる．ネットワークデータの受信処理を図 1 に示す．ネットワークスレッドはイーサネットフレームを受信すると，ネットワークキューにプッシュする．ネットワークキューに含まれるデータは実行スレッドが処理する．promiscuous mode の利用に伴い，ゲスト OS にとって不要なデータがユーザレベルで提供しているバッファにまでコピーされるという問題が生じる．我々は，カーネルレベルでのパケットフィルタ（Pcap Library [9]）を利用することにより，ユーザレベルにコピーする必要がないデータを削除している（図 2 参照）．現在は，仮想 NIC の MAC アドレスに向けられたイーサネットフレーム及びブロードキャストイーサネットフレーム以外を必要のないデータとして扱っている．

仮想ネットワーク機能は TCP/IP を利用してイーサネットフレームを転送する．移動元での受信処理から転送処理への切り換えは次のように実現される．

1. 転送処理へ切り換える前は，移動元のネットワークスレッドが，移動元のネットワークキューに受信データをプッシュする．

2. 移動先の仮想ネットワークスレッドから移動要求が出される．

3. 移動元の仮想ネットワークスレッドが移動要求を受け取り，受信処理を停止するように，ネットワークスレッドに通知する．その後，仮想ネットワークスレッドは移動元のネットワークキューに保存されているデータを転送し，受信したデータを移動先に転送する．

Quasar では，同一サブネットへの移動であった場合，通信元の計算機の仮想ネットワーク機能を無効に

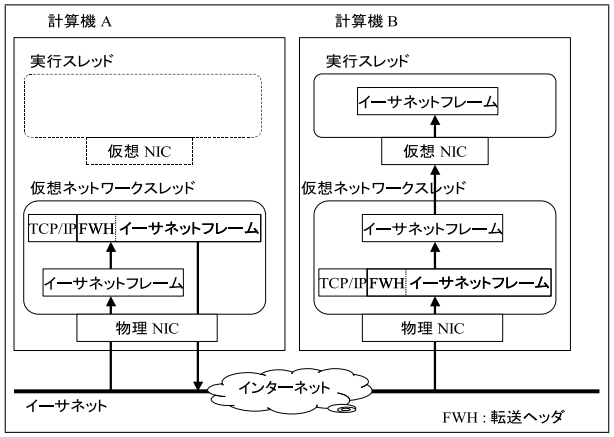

図 3: 受信時の転送処理の流れ（A → B）

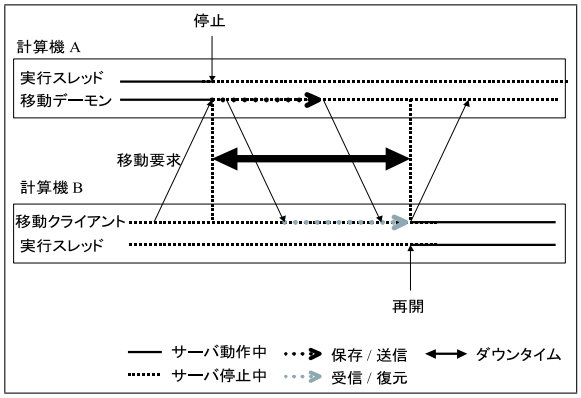

図 4: 一括転送における実行状態の保存・復元処理の流れ

し，通信先のネットワーク機能を有効にする．この判断は移動元の仮想ネットワークスレッドによって行われる．この結果，移動先のネットワークスレッドが直接イーサネットフレームの受信処理を行う．

移動元の物理計算機上で受信したイーサネットフレームの転送処理の流れを図 3 に示す．この転送のための経路は TCP/IP により移動要求時に確立する．移動元のネットワークスレッドは受信したイーサネットフレームに転送ヘッダを加えたデータを転送する．移動先の仮想ネットワークスレッドはこのデータを受信し，転送ヘッダを取り除き，ネットワークキューにプッシュする．

3.4 転送処理のバックグラウンド実行

Quasar では，実行状態を転送する処理をサーバのサービスと並列にバックグラウンドで実行することにより，サービスのダウンタイムを削減する．

単純な実装における実行状態の保存・転送・復元処理の流れおよびダウンタイムとなる時間を図 4 に示す．まず移動元の物理計算機上の Quasar で仮想計算機の実行を停止させてから，実行状態を保存・転送する．移動先の物理計算機上の Quasar は，転送されたデータから実行状態を復元し，復元が終わったら仮想計算機の実行を開始し，サーバのサービスを再開させる．移動元で仮想計算機が停止する瞬間から移動先で仮想計算機が再開する瞬間までは，サービスは利用できない状態になる．

バックグランド処理を用いた実行状態の保存・転送・復元処理の流れおよびダウンタイムとなる時間を図 5 に示す．この処理では保存・転送・復元処理を二段階に分割する．ユーザが移動の指示を出すと，移動元の計算機でメモリとディスクの実行状態の保存・転送を開始する（時刻 1）．この時には，メモリ以外の実行時コンテキスト情報は転送しない．さらに，その間も仮想計算機の実行は継続させる．移動先の計算機では，転送されたメモリとディスクデータの受信・復元処理を開始する．転送中も実行状態は変化し続けているため，転送されるメモリとディスクのデータは，ある瞬間の実行状態を表現しているわけではなく，必ずしも一貫性がない．しかし，後に差分を反映すれば一貫性が取れるので，この時点で一貫性がないことには問題はない．また，この時点では復元した仮想計算機の実行は開始させない．移動先で受信・復元処理が完了したら，再度実行状態の保存・転送要求を発行する．この時点で，移動元は仮想計算機の実行を停止させる（時刻 2）．そして，メモリ以外の実行時コンテキストの実行状態と時刻 1 と時刻 2 の間に加えられたメモリとディスクの実行状態の差分を，移動先に転送する．移動先の計算機は，これらの状態を反映した後，仮想計算機の実行を開始し，サーバのサービスが再開される．

実行状態の転送中に実行状態に加えられた差分を Quasar が認識する方法を述べる．Quasar はメモリの更新されたページとディスクの更新されたセクタをそれぞれ管理するビットマップを持つ．そのビットマップの各ビットは，実行状態の転送中に，そのビットが

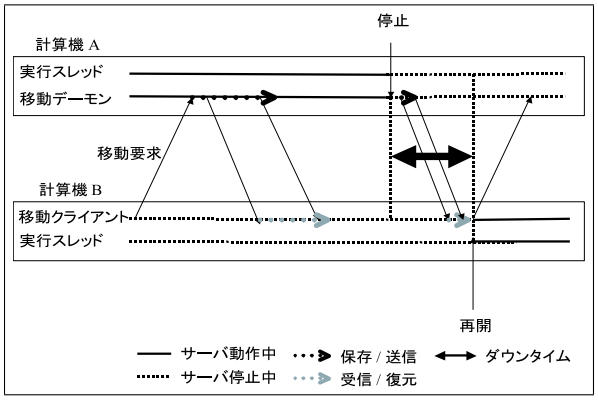

図 5: 分割転送における実行状態の保存・復元処理の流れ

対応する領域に書き込みがあったかどうかを示している．第一段階の転送を行う前にビットマップのビットはすべてクリアされる．メモリへの書き込みを行う仮想計算機の命令を実行する部分に，ビットマップのビットを立てる処理が挿入されている．また，仮想ディスクへの書き込みを行う部分にも，ビットマップのビットを立てる処理が挿入されている．差分を転送する際には，対応するビットが立っている領域のデータのみを転送する．

4 実験

Quasar を実装して実験を行った．実装に用いた QEMU のバージョンは 0.6.0 である．

4.1 実行状態の保存・復元

実行状態を保存したファイルのサイズおよび実行状態の保存と復元に要する時間を測定した．ホスト環境は Linux 2.6.10, Pentium 4 3.0 GHz (Hyper-Threading 有効), 1 GB Memory である．ゲスト OS は Linux 2.4.26 である．ゲスト OS をブートして GUI ログインし，端末エミュレータ kterm を一つだけ実行した．その後，www.google.co.jp の web ページを表示する web ブラウザ Firefox を 10 個起動した．この時点での実行状態を保存・復元した．ゲスト環境のメモリサイズを 128，256，512 MB に変化させて測定を行った．

結果を表 1 に示す．実行状態を保存したファイルのサイズはゲスト環境のメモリサイズにほとんど影響されなかった．保存データのうち，大部分がメモリに関するものである．QEMU のメモリに関する保存・復元処理ではページ単位の圧縮操作が行われ，ゲスト OS のメモリサイズの影響を小さくしている．圧縮操作は，あるページの中身が一様であった場合には 2 バイトで表現される（圧縮されていることを表すためと中身の値を表すためにそれぞれ 1 バイトずつ利用）．保存と復元にかかる時間は，ゲスト環境のメモリサイズの影響を比較的強く受けた．また，保存処理に比べて復元処理は 3 倍程度の時間がかかった．実行状態の保存・復元に要するサイズ，時間ともにある程度小さく抑えられ，有用であると考えられる．

ゲスト環境のメモリ (MB)	実行状態のサイズ (MB)	保存時間 (msec)	復元時間 (msec)
128	127	1089	3708
256	133	1509	4120
512	137	2179	4528

表 1: 実行状態を保存したファイルのサイズおよび実行状態の保存と復元にかかる時間

4.2 仮想計算機上での実行に伴うオーバーヘッド

仮想ネットワーク機能を含む，独自のネットワーク機能を持つ Quasar 上でソフトウェアを実行することによって加わるオーバヘッドの測定を行った．Apache Bench を用いて，ホスト OS 上で直接走る Apache (Apache on host) と，ゲスト OS 上で走る Apache (Apache on guest 1，2) のスループットを比較した．実験に用いた Apache のバージョンは 2.0.52 である．

Apache on host ホスト環境は Pentium 4 3.0 GHz (HT 有効), 1 GB Memory, Linux 2.6.10 である．ただし，測定時には物理メモリ容量を 256 MB に減らし，Apache on guest 1，2 でゲスト環境に割り当てたメモリ容量と同じにした．

Apache on guest 1 ホスト環境は Apache on host と同一のものを用いた．ただし，物理メモリ容量は 1 GM のままにした．ゲスト環境は 256 MB Memory, Linux 2.4.26 である．ここではイーサネット

並列度	1	2	32	128
Apache on host	308.1	468.7	741.5	852.6
Apache on guest 1	65.5	70.5	68.9	68.3
Apache on guest 2	52.8	65.7	65.7	65.4

(Request/Second)

表 2: 負荷 1 に対するスループット

並列度	1	2	32	128
Apache on host	76.4	103.1	111.9	109.1
Apache on guest 1	24.5	23.2	22.7	22.7
Apache on guest 2	20.1	21.4	21.8	20.9

(Request/Second)

表 3: 負荷 2 に対するスループット

フレームを直接受信するネットワーク機能を利用した.

Apache on guest 2 移動先のホスト環境，ゲスト環境は Apache on guest 1 と同一のものを用いた．移動元のホスト環境は Pentium M 1.6 GHz, 504 MB Memory, Linux 2.6.10 である．ここではイーサネットフレームを転送する仮想ネットワーク機能を利用した.

Apache Bench を実行したクライアント計算機は Celeron 700 MHz, 64 MB Memory, Linux 2.4.27 である．本実験ではすべての計算機は同一サブネット内に配置した．このため，3.3 節で述べたように，本来ならば，Apache on guest 2 で利用する 2 台のホスト環境間では転送処理は行われない．しかし，転送処理によるスループットを計測するため，この実験では擬似的に転送処理を行うように設定した.

Apache Bench の負荷として以下の 2 つを用いた.

負荷 1 リクエストされるファイルのサイズは 10 KB, リクエストの数は 1024

負荷 2 リクエストされるファイルのサイズは 100 KB, リクエストの数は 1024

Apache Bench のリクエストの並列度を 1，2，32，128 に変化させて測定を行った.

結果を表 2，3 に示す．Quasar 上で実行することにより，負荷 1，2 それぞれに対するスループットは 1/5 から 1/10，1/3 から 1/5 程度に低下した．本実験では LAN 環境で実験したので，仮想計算機の実行とネットワーク処理によるオーバヘッドが際立つ結果となっている．実際の運用ではネットワーク遅延が存在するため，スループットはこの結果ほどは下がらないと予想される.

4.3 ダウンタイムの削減

ダウンタイム削減のための最適化の効果を調査した．実行状態を 1 回で転送する場合と，バックグラウンド転送を利用して 2 回に分けて転送する場合とで，システム停止時間を測定した．実験には以下の 2 台の計算機を用いた.

計算機 A Pentium 4 3.0 GHz (HT 有効), 1 GB Memory, Linux 2.6.10

計算機 B PowerPC 450 MHz, 384 MB Memory, Linux 2.6.10

2 台は同一サブネット内に配置した．ゲスト環境は 128 MB Memory, Linux 2.4.26 である．まず，片方の計算機上でゲスト OS をブートし，GUI ログインする．ログインしたら，端末エミュレータ kterm を実行し，その中でコマンド top を実行する．そして，top を実行したまま，仮想計算機を他方の物理計算機に移動する．次の 2 つの時間を計測した.

- 実行状態を 1 回で転送する方式において，転送元で仮想計算機の実行が停止してから，再開完了の通知を受けるまでの時間（一括転送時停止時間）
- 実行状態を 2 回に分けて転送する方式において:
  - 転送元で実行状態の転送が開始してから，再開完了の通知を受けるまでの時間（二段階転送時再開時間）
  - 転送元で仮想計算機が停止してから，再開完了の通知を受けるまでの時間（二段階転送時停止時間）

計算機 A から計算機 B への移動を行う場合と，計算機 B から計算機 A への移動を行う場合の両方で測定を行った.

結果を表 4 に示す．実行状態を 1 回で転送する方式では 8 秒以上の停止時間が生じたが，2 回に分けて転

	一括転送時 停止時間	二段階転送時 再開時間	二段階転送時 停止時間
A → B	8348	8542	226
B → A	10450	11248	317

(msec)

表 4: 仮想計算機の移動に伴うダウンタイム

送する方式では停止時間は数百ミリ秒程度に抑えることができた．また，移動先での再開時間は，実行状態を 1 回で転送しても 2 回で転送しても，ほとんど変化がなかった．この実験では，分割転送方式を利用することでダウンタイムを大きく削減することができた．

5 関連研究

仮想計算環境を提供するシステムをホスティングや移動計算に応用する研究は，これまでに非常に多数存在する．VMotion [13] は VMware ESX Server に移動機能を付加したシステムである．本研究ではダウンタイムの削減の際にビットマップを利用する部分に VMotion の技術を適用している．VMotion ではサブネット内の移動を想定している．また，Storage Area Network (SAN) の利用により，仮想ディスクの転送は行われない．我々のアプローチでは異なるサブネットへの移動もサポートし，仮想ディスクの転送を行う．Collective [10] では，VMware の仮想ディスクの転送により計算の移動を実現する文脈において，仮想ディスクの圧縮や差分転送などの多くの最適化を提案している．文献 [14] の研究では，遠隔に保存された VMware の仮想ディスクと利用して仮想計算機を生成する文脈において，仮想計算機を高速に起動し，効率的に実行するための分散ファイルシステムサポートを提案している．Internet Suspend/Resume [17] は VMware の仮想ディスクと分散ファイルシステムを利用し仮想計算機を復元する仕組みを提案している．我々のアプローチでは仮想ディスクの転送を行い，仮想ディスクへのアクセスは常にローカルアクセスとして行われる．VMware を基盤に用いたシステムでは，利用可能なゲスト CPU とホスト CPU が IA-32 だけに制限される．Quasar は CPU エミュレータを基盤にしているため，多様なゲスト CPU とホスト CPU を用いることができる．VMware を基盤に用いたシステムの研究において提案された要素技術の多くは Quasar にも適用可能である．

システムコールの仮想化や資源ビューの仮想化などの OS レベル仮想化を用いた仮想計算環境システムを基盤に用いてホスティングや移動計算を実現するアプローチも多く提案されている．たとえば，Zap [2], MobiDesk [18]，SBUML [15], SoftwarePot [1, 3], Compute capsule [11] などがある．これらのアプローチは，CPU エミュレータを用いるアプローチに比べ，オーバヘッドを小さく抑えられる可能性があるが，OS や CPU が異なる計算機間で仮想計算環境を移動させる用途には適さない．

Quasar に実装されている移動を意識したネットワーク機構それ自身は新規なものではない．アプリケーションに対して透明な形での計算機の移動を可能にするための既存研究には，たとえば MobileIP [19, 20]，M-TCP [12]，文献 [16]，VNAT [21] がある．これらのアプローチではネットワーク層やトランスポート層で通信処理を行っている．我々のアプローチではデータリンク層のイーサネットフレームを用いることで，ゲスト OS が DHCP サーバから IP アドレスを取得し，外部通信を行うことが可能である．Quasar がゲスト OS の IP アドレスを認識する必要がない．

6 まとめと今後の課題

CPU エミュレータ QEMU を用いて仮想計算機の移動を支援する機能を備えたシステム Quasar の設計と実装を行った．Quasar を用いて実験を行った結果，実行状態の保存・復元に要するサイズ，時間ともある程度小さく抑えることができた．また，システム停止時間を大きく削減することができた．しかし，仮想計算機上での Apache のスループットの低下は大きかった．

今後の課題としては，Quasar 上で動作するソフトウェアのスループットを向上させることが挙げられる．ネットワーク機能を含めた仮想化処理の詳細な解析を行い，Quasar を改善していきたい．また，分割転送方式では 1 回目の処理中のシステムの動作に依存するため，システム停止時間が増加する場合もある．これを改善するために，demand-driven で実行状態を転送する方法を導入したい．現在，仮想ディスクの移動には差分ディスクを利用しているが，複数の物理計算機上の仮想ディスク間の整合性の維持やサイズの大きい初

期データを含む仮想ディスクの効率的な転送が必要となる．これらの仮想ディスクの制御を行うための枠組みの導入をしたい．さらに Quasar を負荷分散や耐故障の基盤システムとして発展させていくことも考えている．これを実現するため，負荷の増加や障害を自律的に検知して仮想計算機を移動・複製する処理を実装したい．

謝辞

多くの有益なアドバイスをいただいた東京大学米澤研究室の前田俊行氏，金田憲二氏に感謝の意を表す．

参考文献

[1] K. Kato and Y. Oyama. SoftwarePot: An Encapsulated Transferable File System for Secure Software Circulation. In *Software Security – Theories and Systems*, Vol. 2609 of *Lecture Notes in Computer Science*, pp. 112–132, February 2003.

[2] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The Design and Implementation of Zap: A System for Migrating Computing Environments. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, December 2002.

[3] S. Péter, H. Abe, T. Hirotsu, Y. Shinjo, and K. Kato. General Virtual Hosting via Lightweight User-mode Virtualization. In *Proceedings of the 2005 International Symposium on Applications and the Internet (SAINT 2005)*, January-February 2005.

[4] QEMU CPU Emulator. http://fabrice.bellard.free.fr/qemu/.

[5] Java. http://java.sun.com/.

[6] Microsoft. .NET. http://www.microsoft.com/net/.

[7] bochs. bochs. http://bochs.sourceforge.net/.

[8] VMware. VMware. http://www.vmware.com/.

[9] tcpdump/libpcap. http://www.tcpdump.org/.

[10] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum. Optimizing the Migration of Virtual Computers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, December 2002.

[11] B. K. Schmidt, M. S. Lam, and J. D. Northcutt. *Supporting Ubiquitous Computing with Stateless Consoles and Computation Caches*. PhD thesis, Computer Science Department, Stanford University, August 2000.

[12] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode. Migratory TCP: Connection Migration for Service Continuity in the Internet. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS '02)*, pp. 469–470, July 2002.

[13] VMware. VMotion. http://www.vmware.com/products/vmanage/.

[14] M. Zhao, J. Zhang, and R. Figueiredo. Distributed File System Support for Virtual Machines in Grid Computing. In *Proceedings of the HPDC-13*, pp. 202–211, June 2004.

[15] O. Sato, R. Potter, M. Yamamoto and M. Hagiya. UML Scrapbook and Realization of Snapshot Programming Environment. In *Software Security – Theories and Systems, Second Mext-NFS-JSPS International Symposium, ISSS 2003*, Lecture Notes in Computer Science, Vol. 3233, pp. 281-295, Tokyo, Japan, November 2003.

[16] F. Teraoka, Y. Yokote, M. Tokoro. A Network Architecture Providing Host Migration Transparency. In *Proceedings of ACM SIGCOM 91*, pp. 209–220, September, 1991.

[17] M. Kozuch and M. Satyanarayanan. A Internet Suspend/Resume. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, Callicoon, NY, June, 2002.

[18] R. Baratto, S. Potter, G. Su, and J. Nieh. MobiDesk: Mobile Virtual Desktop Computing. In *Proceedings of the 10th Annual ACM International Conference on Mobile Computing and Network (Mobicom 2004)*, Philadelphia, September-October, 2004.

[19] C. Perkins. IP Mobility Support for IPv4. IETF RFC 2002, October, 1996.

[20] D. Johnson, C. Perkins. Mobility Support in IPv6. In *Proceedings of the 2nd Annual ACM International Conference on Mobile Computing and Network*, Rye, New York, November, 1996.

[21] G. Su, J. Nieh. Mobile Communication with Virtual Network Address Translation. Technical Report CUCS-003-02, Department of Computer Science, Columbia University, February, 2002.