2006－OS－103　(16)
2006／8／1

仮想計算環境の効率的な転送法に関する研究

川崎　仁嗣†，阿部　洋丈‡，Richard　Potter‡，加藤　和彦†

† 筑波大学　　‡ 科学技術振興機構

仮想計算機技術を利用した仮想計算環境をネットワーク上で効率的に転送する方法を提案する。仮想計算環境を転送する際に保存すべきデータは、大きく分けてファイルシステムとメモリイメージの 2 つである。本稿では後者に注目し、メモリイメージ中の不用なデータを破棄することにより保存すべきデータの量を減らした。メモリページを管理する構造体へアクセスし、不要なデータを判別した。

A study on efficient transfer method for virtual computation environment

Satoshi Kawasaki†, Hirotake Abe‡, Richard Potter‡ and Kazuhiko Kato†

† University of Tsukuba　　‡ Japan Science and Technology Agency

We proposed the efficient method for transferring virtual machine environment over network. For tranferring virtual machine environment, there are two types of data to save, file systems and memory images. We focus attention on latter in this study and reduce size of data to save by discarding disused data in memory images. To discriminant which data is disused, this method accesss to memory page descriptor table.

1．はじめに

仮想計算環境ではメモリやファイルシステムなどが仮想化されるため、その時点での動作状態をスナップショットという形でファイルなどに保存しておくことも可能である。したがって、スナップショットを複数保存しておくことにより、任意の時点まで動作状態を戻すことが簡単に出来る。スナップショットを複数の計算機へ配布することにより仮想計算環境のコピーを簡単に作ることが出来る。特定のサービスを仮想計算環境上で実行し、その動作状態を簡単に他の計算機上でも再現できる。これにより、もしサービス提供が不可能になってしまっても他の計算機上でサービスの提供を継続することが簡単に行える。これに関する研究として、サステーナブルサービス[6]がある。また、継続的なサービス可用性を維持したまま、ホストで実行中の仮想マシンを別のホストに移動する手法としてVMotion[8]がある。

本稿では、このスナップショットをネットワーク上でより効率的な転送が行える方法を提案する。効率的な転送にはスナップショットのサイズを小さくすることが重要となってくる。既存手法を利用することで、ファイルシステムについてはかなりのサイズ削減が実現できている。しかし、メモリイメージについてはスナップショット間での差分をとる以外のサイズ削減方法はなかった。現在の一般的環境では、メモリサイズが数百 MB ある環境も稀ではない。そこで、本稿ではメモリイメージでの利用されていないデータまでがスナップショットに保存されてしまっていることに着目した。今後、利用されることのないデータをスナップショットに保存しなければならない必要はないので、これらのデータを破棄することによりスナップショットサイズの削減が期待できる。また、提案手法の有効性を明らかにするために、我々の提案する手法を実装しスナップショットのサイズを比較した。実験結果より、提案手法は仮想計算環境の効率的な転送に有効であることを確認した。

2. 準備

本稿の提案手法の説明のために、いくつかの用語について述べる。まず、我々の用いた仮想計算環境について説明する。また、本研究と関わりのあるいくつかの機能について述べる。

2.1. User Mode Linux

仮想計算環境を実現する方法にはいくつか種類があり、大きく分けると (1) 完全仮想化（full virtualization）、(2) 準仮想化（para-virtualization）、(3) OS 環境の仮想化 の 3 つに分けることが出来る。(3)を利用した仮想計算環境として User Mode Linux（UML）[2][4]がある。

UML は Linux カーネルをユーザーモードプログラムとして実行できるように拡張を行ったものである。UML を動作させる OS 環境をホスト OS と呼ぶ。これに対し、ホスト OS 上で実行される OS をゲスト OS と呼ぶ。これを用いるとホスト OS の内部で複数の仮想的な OS を実行させることができる。最新のカーネルでは UML の拡張がソースコードツリーにマージされており、もはやカーネルの一機能として扱われている。UML の拡張のほとんどは、アーキテクチャ依存コード(arch/um 以下)として記述されている。したがって、プロセス管理やメモリ管理など OS の主要な機構は一般的なカーネルと同一のものである。

仮想化されたメモリは、ホスト OS 上のファイルとして構成される。具体的には、以下に示す 4 つのファイルが生成される。

(1) vm-1 カーネル実行形式の text 領域
(2) vm-2 カーネル実行形式の data 領域
(3) vm-3 カーネル実行形式の bss 領域
(4) vm-4 ユーザプロセス用として用いられる領域

vm-1、vm-2、vm-3 は UML 内で実行されるプロセスにおいて共通である。vm-4 ファイルは、UML のページサイズごとに分割され UML のページング機構によってユーザプロセスへ割り当てられる。アドレス空間としては、図 1 に示したようになっている。

メモリの仮想化と同様に、ファイルシステムもホスト OS のファイル上に仮想ディスクとして構成されている。UML には IO thread というカーネルスレッドがあり仮想ディスクへの読み書きなどが行われる。

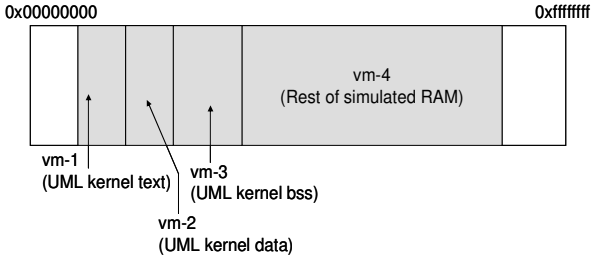

図 1 UML のメモリアドレス空間
（[1]より引用）

2.2. SBUML（Scrap Book for User Mode Linux）

SBUML[1][5]とは、User Mode Linux に対してサスペンドやスナップショットの保存・復元や状態抽出、実行制御などができるように拡張したものである。スナップショットとは仮想計算環境の動作状態をファイルとして保存したものである。

UML の状態保存

UML の状態保存を行うために、保存すべき情報は大きく分けて 4 つある。

(1) UML 仮想メモリ
UML の仮想メモリは、前述したように vm-1、vm-2、vm-3、vm-4 の 4 つのファイルに保存されているため、このファイルを保存することで仮想メモリの状態保存が行える。

(2) UML ファイルシステム
UML のファイルシステムはホスト OS 上のファイルに仮想ディスクとして実装されているため、このファイルを保存すればファイルシステムの状態保存が行える。

(3) UML プロセスの状態
プロセスの状態には以下の 2 つがある。
- プロセスアドレス空間の復元

プロセスアドレス空間には vm-4 ファイルがページサイズ単位で細切れに割り当てられている。したがって UML カーネル内のマッピング情報を保存すればよい。

• 「ホストプロセス」としてのコンテクスト
ホスト側におけるプロセスのレジスタ値などの実行コンテクストを保存する。

(4) UMLが使用しているファイルディスクリプタに関する情報
仮想メモリ、ファイルシステム、仮想端末用のxterm ウィンドウなどの、UMLがホストOS上で使用しているファイルディスクリプタを保存する。

基本機能

(1) サスペンド、レジューム機能
スナップショットを保存している途中で、各UMLプロセスがメモリやファイルシステムにアクセスしないよう、全ての UML プロセスをブロックする必要がある。サスペンドすると各 UML プロセスは一定箇所でループする。

(2) スナップショットの保存
スナップショットの保存は上で述べた 4 つの情報をコピーして保存することで行える。しかし、UML プロセスを実行している状態でコピーを行うとデータの不整合が発生する可能性がある。そのため、サスペンドしてからファイルをコピーする。スナップショットファイルについて、図 2 に示す。

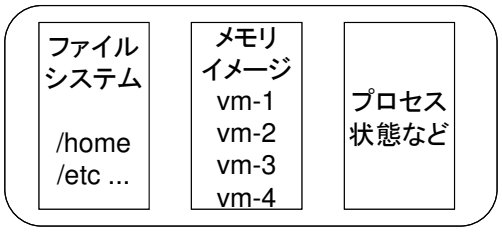

図 2 スナップショットファイル

(3) スナップショットからのUML再開
スナップショットからの再開では保存していた情報を元通りに再配置し、必要なプロセスやスレッドを生成することにより行われる。

2.3. Linuxカーネルにおけるメモリ管理

ここでは、UMLのベースであるLinuxカーネルのメモリ管理方法について説明する。UMLやSBUMLにおいてもメモリ管理方法はLinuxカーネルのものとほぼ同等である。ただし、割り当てられるメモリは実際にはホストOS上のファイルにマッピングされている。

全てのメモリは固定サイズで分割したページフレームの集まりとみなされる。ページフレーム内にはページをおくことができ、ページフレームのサイズはページのサイズと同じである。ページとは、実際のデータをページフレームサイズで分割した、データの集まりである。

ページディスクリプタ

カーネルは各ページフレームを管理するために、page 構造体として定義されたページディスクリプタにページフレームの情報を保持している。保持している情報としては、そのページフレームがプロセス用のページ、カーネルコード、カーネルデータ用のページのいずれを含んでいるかの識別情報や、空きページフレームかどうかを示す情報などがある。

メモリゾーン

ページフレームには、コンピュータアーキテクチャの制約により、ページフレームの用途に制約がついている。また、32 ビットコンピュータでは 896MB 境界を越えたところにあるページフレームに直接アクセスすることは出来ない。これはリニアアドレス空間が小さすぎるためである。これら 2 つの制約のため、Linuxでは物理メモリを次の 3 つのゾーンに分けて管理している。(1) ZONE_DMA、(2) ZONE_NORMAL、(3) ZONE_HIGHMEM である。SBUML の場合、高位メモリ（HighMem）を利用しなければ(1)のみが利用される。

表 1 ゾーンディスクリプタ

名前	役割
size	ゾーン内のページ数
free_pages	ゾーン内の空きページ数
free_area	開放されたページフレームのサイズ別リスト
zone_mem_map	ゾーン用のページディスクリプタの配列
zone_start_paddr	ゾーンの先頭物理アドレス

カーネルはこれらのゾーンを管理するため、zone_struct 構造体として定義されたゾーンディスクリプタにゾーンの状態を保持している。zone_struct 構造体の内、主要なメンバを表 1 に示す。

3. 既存手法

仮想計算環境の効率的な転送のためには、スナップショットのサイズを可能な限り小さくする必要があると考えられる。現在の SBUML では以下に示す 2 つの方法によりスナップショットのサイズを削減している。

3.1. ベースイメージの利用

スナップショットには 2.2.UML の状態保存(2) で述べたとおり、ファイルシステムやメモリイメージが含まれる。それらをそのままスナップショットに保存すると非常に大きなファイルとなってしまう。これをネットワーク越しで転送するのは容易ではない。一般的な使用用途であれば、ファイルシステム全体に変更が及ぶことはまずない。したがって、ファイルシステムに関しては基本となるファイルシステムを用意しておく。これをベースイメージと呼ぶ。そして、ベースイメージとの差分のみを保存することで、大幅にスナップショットサイズを小さくすることが出来る。ベースイメージとの差分をとるには UML の COW（Copy on Write）ファイルシステムを利用している。

COW ファイルシステムでは、ファイルシステムからの読み込みはベースイメージのデータを読み込む。しかし、ファイルシステムへの書き込みが行われるときは、ベースイメージから書き込み対象のブロックのデータを別に用意したファイル（COW ファイル）へコピーし、書き込みが行われる。それ以後は、書き込みの行われた領域の読み込みが行われると COW ファイルからデータが読み込まれる。この様子を図 3 に示す。

COW ファイルは元のベースイメージと理論的には同じファイルサイズである。しかし、ファイルの中で変更を加えていない部分は 0 であり、スパースである。Linux ではスパースなファイルを効率的に扱うことが出来る。具体的には、実際にデータの書きこまれているブロックのみをディスク上に保存している。したがって、COW ファイルの実際のサイズはベースイメージからの変更差分のみになる。

なお、UML のメモリイメージも同様にスパースとして処理されている。

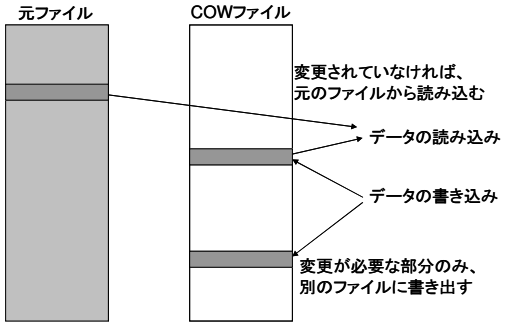

図 3 COW ファイルシステム

3.2. スナップショット間の差分の利用

スナップショットを複数とるような場合には 3.1 で示したような方法に加えて、前回のスナップショットとの変更差分をとることによりさらにスナップショットのサイズを削減することが出来る。この方法では、前回のスナップショットをとった時点から変更されていないファイルやメモリイメージを省くことが出来る。この様子を図 4 に示す。 SBUML では、xdelta[7]を利用して差分を生成している。

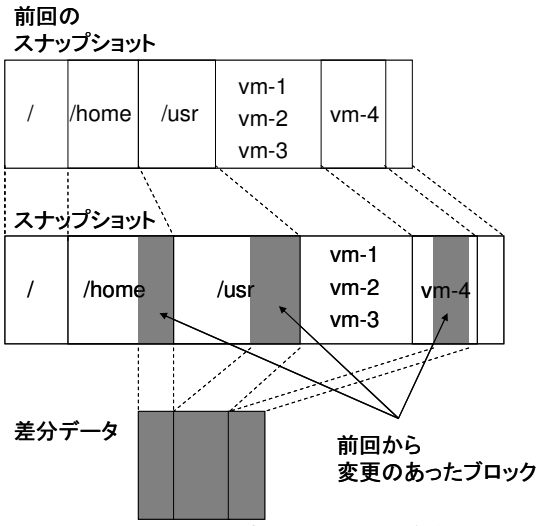

図 4 スナップショット間の差分

4. 提案手法

現在のほとんどの OS は性能上の理由から、メモリ領域を開放するとメモリの管理テーブルに未使用のマークを付けるだけで、メモリ領域上の実際のデータは消去されない。したがって、利用されていないメモリ領域にもかかわらず実際にはメモリ上にデータが存在することがある。そして、このメモリ領域をユーザモードプロセスへ割り当てる必要が生じた際にメモリ領域の初期化が行われる[1]。しかし、スナップショットを保存する場合はデータが 0 でないとスパースとしては扱われないため、不用なデータにもかかわらず保存されてしまっている。

われわれの手法では OS の持つメモリ管理テーブルを読み取り、実際にはメモリ上にデータが存在する領域を消去する。利用されていないメモリ領域を見つけるためには、メモリ管理テーブルの該当するページの参照カウンタの値が 0 であるかどうかをチェックしている。参照カウンタとは、そのメモリ領域が割り当てられるたびに 1 ずつ増やされ、開放されると 1 ずつ減らされる。したがって、この参照カウンタの値が 0 であれば対応するメモリ領域は利用されていないことになる。利用されていない領域であるが、ページがダーティ（使用済みデータが残っている）であれば、そのページ全体を 0 クリアする（図 5）。スナップショットを保存する直前にこのような操作を行うことでそれらのページはスパースな領域として扱われるので、今後利用することのないデータが含まれているメモリ領域は保存されないようになる（図 6）。また、保存するべきデータの量が削減されているので、スナップショットの保存や復元処理の高速化も期待できる。

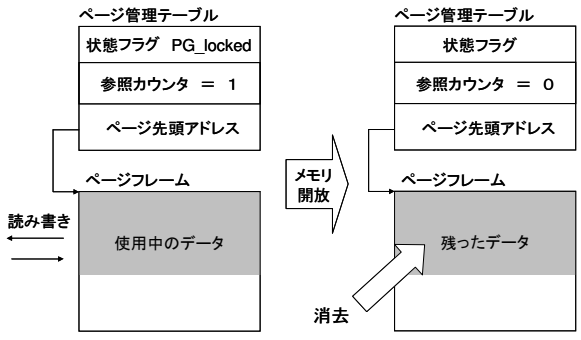

図 5 不用なデータ

[1] カーネルが利用する場合、初期化は行われない。

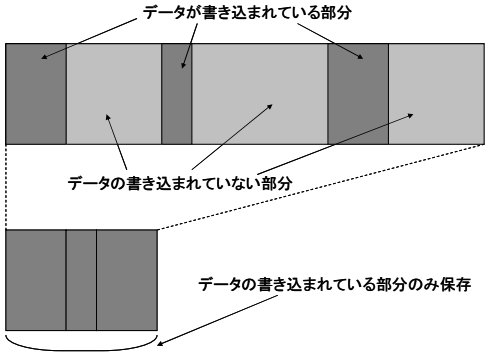

図 6 スパースファイル

5. 実験

5.1. 実験概要

本稿では、実装を行う仮想計算環境として SBUML を用いた。SBUML の実装はアーキテクチャ依存コードとして実装されているため、メモリ管理を行うコードの大部分は Linux カーネルのものと同等である。カーネルは全てのメモリ領域を固定サイズ（i386 では 4kB）ごとに区切ったそれぞれのページフレームを管理するため、ページディスクリプタという構造体に管理情報を書き込んでおり、それらは配列として保存されている。ページディスクリプタには参照カウンタとして、count というメンバが存在する。したがってこの値だけをチェックすれば良いように思えるが、複数ページ（ブロック）をまとめて割り当てる際にはそれらのページの内、先頭ページの参照カウンタしか更新されない。

メモリゾーンにおける空きページフレームは、ゾーンディスクリプタの free_area メンバで管理されている。free_area メンバには 10 個の要素があり、それぞれはブロックのサイズ（ページ数）が 2^0～2^9 の空きブロックの情報を管理している。具体的には、free_area メンバの要素が持っている空きブロックリスト free_list に空きブロックの先頭ページディスクリプタがリストとしてつながれている。この様子を図 7 に示す。

したがってページディスクリプタを順にたどり、そのページフレームがダーティである場合には、そのページを 0 で初期化する。すべてのページにおいてこれらの操作が終了すればメモリ（実際にはメモリイメー

ジファイル）内の不用なデータを削減できたことになる。その後、スナップショットの保存を行う。

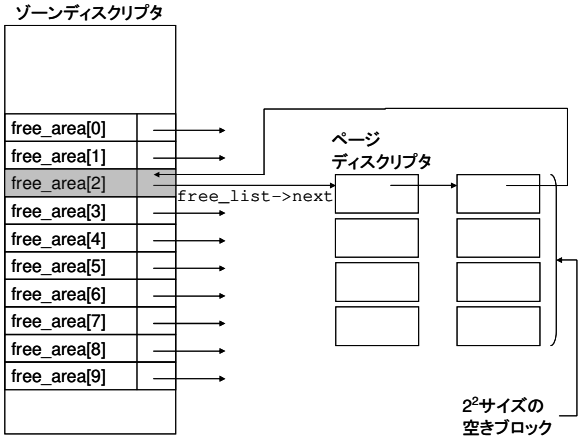

図 7　空きブロックの管理

VNC を終了させた。左は提案手法なし、右は提案手法ありの場合である。

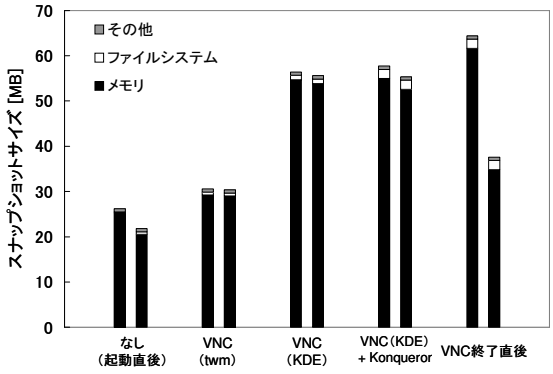

図 8　実験結果（メモリ 64MB）
左：提案手法なし　右：提案手法あり

5.2. 実験手順

メモリイメージのサイズ削減処理に要する時間、およびスナップショットの保存、復元にかかる時間を測定するために以下の手順で実験を行った。ゲスト OS として Redhat Linux 7.3 を利用し、ゲストへ割り当てるメモリサイズを 64MB、256MB と変化させて実験を行った。

1. （ホスト上で）SBUML をブートする。
2. メモリ領域にデータを載せるため、SBUML 上でプログラムを動作させる。
3. メモリイメージのサイズ削減処理を行う。
4. （ホスト上で）SBUML のスナップショットを保存する。
5. 4 の操作で生成されたスナップショットのファイルサイズを測定した。

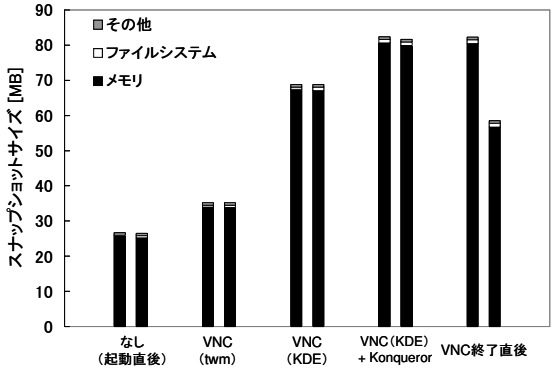

図 9　実験結果（メモリ 256MB）
左：提案手法なし　右：提案手法あり

5.3. 実験結果

スナップショットファイルサイズ

スナップショットのファイルサイズについて測定した結果を以下に示す。なお、図 8 はメモリサイズ 64MB における結果、図 9 はメモリサイズ 256MB における結果である。それぞれのプログラムを起動した後、スナップショットを保存した。具体的には SBUML 起動後に、VNC（twm）を実行した場合、VNC（KDE）を実行した場合の 2 通りである。VNC（KDE）上ではさらに Konqueror（ブラウザ）を起動し、その後

5.4. 考察

図 8、図 9 より、プログラムの動作中は不必要なメモリがほとんどなくあまり提案手法の効果が現れない。しかし、プログラムを終了させるとプログラムに使用していたメモリが解放されるため、今回の提案手法を行うとスナップショットのサイズを 2 割～4 割程度削減することが出来た。もし何もせずに普通にスナップショットを保存すると、これらの不用データ分だけスナップショットは大きくなってしまう。

図 9 では、図 8 と比較してそこまで削減量が多くない。これはメモリに余裕があるとカーネルはディスクキャッシュを大きくとるためである。したがってキャッシュに割り当てられている領域分は削減出来ない。

以上より、プログラムの起動、終了を繰り返すような場合に今回の提案手法の効果が大きいことが分かった。したがって、メモリの確保、解放を繰り返すようなプログラムでも効果があると思われる。

6. 関連研究

仮想計算環境の状態保存ファイルのサイズを削減する手法で、今回の提案手法と類似した手法として Ballooning [3][9]がある。この手法では、ゲスト OS に大量のメモリ割り当てを要求し確保できた領域を初期化している。アプローチしては本研究と類似している。この方法ではカーネルの持つメモリ管理機能によりメモリを確保するため、最小限必要なデータ以外は消去してしまうことができる。ただし、メモリ確保の過程でスワップアウトが発生してしまうのでゲスト OS の性能が落ちてしまう可能性がある。

7. まとめ

本稿では、仮想計算環境をネットワーク上でより効率的に転送する手法を提案した。我々の手法では、メモリイメージに含まれる使用済みで今後利用されることのないデータを消去してしまうことにより、メモリイメージのサイズを削減することが可能である。これにより、必然的にスナップショットファイルのサイズも削減することが可能である。そして、スナップショットのサイズが小さくなることでネットワークを介した転送をより効率的にすることができる。

今後の課題としては、カーネル内で、まだ利用される可能性があるためにキャッシュに残されているデータについても消去できるように改善を行っていく予定である。

謝辞

本稿での実験実装について、筑波大学システム情報工学研究科 榮樂英樹氏に多くの助言をいただきました。ここに深く感謝いたします。

参考文献

[1] O.Sato, R.Potter, M.Yamamoto and M.Hagiya.： “UML Scrapbook and Realization of Snapshot Programming Environment”, Proc. of the International Symposium on Software Security 2003.

[2] The User-mode Linux Kernel . http://user-mode-linux.sourceforge.net/

[3] C. A. Waldspurger. ： "Memory resource management in VMware ESX server.", In Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI), December 2002.

[4] Jeff Dike ： "A user-mode port of the Linux kernel", the 4th Annual Linux Showcase & Conference, 2000.

[5] Richard Potter： One-Click Distribution of Preconfigured Linux Runtime State, USENIX Virtual Machine Research & Technology Symposium (VM), 2004.

[6] 小磯知之, 阿部洋丈, 鈴木与範, Richard Potter, 池嶋俊，加藤和彦 ：サステーナブルサービスを実現する基盤ソフトウェアの設計，SACSIS 2006

[7] xdelta project http://sourceforge.net/projects/xdelta/

[8] Michael Nelson, Beng-Hong Lim, and Greg Hutchins ： Fast Transparent Migration for Virtual Machines, USENIX 2005

[9] Constantine P. Sapuntzakis, Ramesh Chandra, Ben Pfaff, Jim Chow, Monica S. Lam, Mendel Rosenblum ： Optimizing the Migration of Virtual Computers, OSDI 2002