仮想環境下におけるゲストOS間共有ライブラリーの構築手法

坂下善彦[†1]　　中尾司ピエール[†2]

同一のあるいは類似のOSが備えている共通のライブラリーを，KSMの機構とは別に，メモリ管理機構において管理している当該共通ライブラリーの格納されている場所データを，ホストOSが司るメモリ管理機構が管理する共通領域に射影させてre-entrantな共通ライブラリーとして管理する手法を検討している．
LD_LIBRARY_PATH環境変数と関連する管理データ(プログラムの配置に関するmm_struct, プロセスの配置に関するvm_area_struct，等)に注目して，仮想計算機の間で連携して実現する方式と，仮想計算機とホスト間で連携する方式を想定している．これらに基づく構築手法の検討状況を報告する．

Proposed Construction Method of guest OS between shared libraries in a virtual environment

Yoshihiko Sakashita[†1]　　TsukasaPierre Nakao[†2]

1. はじめに

CPU 関連のハードウェアアーキテクチャの近年における進歩により，コンピュータシステムで実施できる機能が格段に向上している．特に，最近の仮想化機構の発展は目覚ましい．これに相俟って，オペレーティングシステムの対象とする機能や処理がより高度に発展し，さまざまな機能やサービスを提供できるようになった．これらの技術を駆使した所謂クラウドシステムの特に IaaS の側面における機能の充実が著しい[7].

我々は，このような中でLinux仮想環境におけるシステムのメモリ使用量を軽減させる仕組みである Kernel-based Virtual Machine(KVM)とメモリ使用の状況をメモリコミットの観点から見てきた[1]. KSMは，複数ゲストOSが管理するメモリを横断的に見て，ページ単位で同じデータが存在する場合は１つのデータ源としてまとめて，CoW(Copy on Write)の状態で管理する手法である[2][3][14].

この手法は，確率的に類似のデータが存在する高い場合に有効であり，具体的には，同一あるいは類似のプログラムが複数ゲストOS上で稼働する場合や，同一のデータをそれぞれのゲストOS上のプログラムが使用する場合などに，効果が期待できる．

見方を変えると，ゲストOSが同一の場合は，OSのコード自身が類似のデータとなり，OS 上のアプリケーション以外の対象要素となる．我々の実験においても，かなりの割合で圧縮されている現象を確認している．

そこで我々は，同一のあるいは類似の OS が備えている共通のライブラリーを，本 KSM の機構とは別に，システムの共通領域に存在させて re-entrant な共通ライブラリーとして管理する手法を検討している．

例えば，同一のOSであれば，OSシステムが抱えているシステムライブラリーの類は共通で使用できるので，これらを，共通領域（仮想機械の場合では，ホストが管理する匿名領域のデータおよびスタック域以外の場所)に配置する．この場合，メモリ管理が論理的には重構造になる．即ち，ゲストOSのメモリ管理機構において管理している当該共通ライブラリーの格納されている場所データを,ホストOSが司るメモリ管理機構が管理する場所データへ射影する．このような機構を目的としている．

これにより，更に，ゲストOS上で稼働するアプリケーション間で共通するライブラリーがある場合は，同様の機構を通して共通化することが可能になると考えている．

2. 仮想計算機における資源

KVM により構築される仮想化計算機環境において，各仮想計算機（VM）を，CPU，メモリ，ハードディスク，その他の入出力などの資源などシステム全体を再現することにより実現している[2]. 仮想計算機は資源を利用する上で仮想計算機を実現する側（ホスト）のハードウェアをハイパーバイザが効率よく配分することで実現している．このように，仮想計算機は単一の実計算機（ホスト）上で複数の仮想計算機が稼働されている状況を作り出している．

各仮想計算機が要求する資源を割り当てる上で必要なメモリは静的に割り当てられる．このために，仮想計算機の数だけのメモリ資源が提供されることになるが，しかし，各仮想計算機は要求したメモリ資源をすべて使用している状

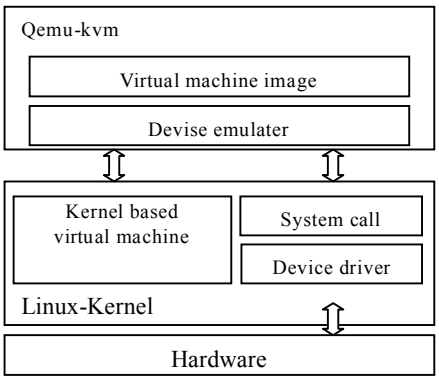
†1 湘南工科大学工学部情報工学科
Shonan Institute of Technology, Information Sience Dep.
†2 湘南工科大学工学研究科電気情報専攻
Shonan Institute of Technology, Graduate school,

況の場合はかなりの確率でメモリスワップが発生してしまうと考えられるが，現実には極めて少ないと予想される．
このように実行時の要領を精度よく見積もって要求する容量を申請することは実際には難しい．他方，計算機システムを管理運用する側からは，この見積もり量の差分は全体的な性能劣化を起こす要因の一つともなり，避けたいものである．
このように，仮想計算機環境を多くのユーザが使用する場合，あるいは比較的多くの資源容量を要求する場合には計算機自体の資源の容量との関係から，ユーザの利用状況を妨げることなくシステム全体として資源に余裕があることが望ましい．このことは，クラウドコンピューターシステムにおける資源見積りと同様の課題である．

3. メモリ消費と KSM

3.1 メモリ使用量

このように構築される仮想化計算機環境において，ホスト OS の Linux カーネルによって提供される Kernel Samepage Merging(KSM)を利用することでホスト OS のメモリが節約できることが分かってきた[1][2].
図 1 に示すように，Linux KVM は KVM カーネルモジュールとユーザモードで動作する qemu-kvm とにより構成され，ユーザモードのプロセス要求に応じてカーネルモジュールは，仮想計算機にメモリを割当てて実行する機能を備えている．



Pic.1 Virtual Computer

KSM は，ユーザプロセスの匿名ページ領域をスキャンし，同一内容のページが存在する場合は，１つのページにマージすることでメモリの使用量を抑えている．
カーネルスレッド ksmd が指定された時間間隔で指定ページを memcmp を用いてスキャンし，マージされたページは CoW(copy on write)の状態で管理される．このような処理の結果実際に使用されるメモリの容量が圧縮されることになる．この圧縮量が大きくなることを期待することになる．
我々の実験では，意図的にメモリ資源を確保してそこに同一データを書き込ん場合，ランダムな値のデータを書き込んだ場合におけるこのマージの状況を観察し確認している．
しかしながら，現実の利用状況におけるデータの形態を予想することは極めて困難であり，使用するアプリケーションによってその実態はさまざまであると予想される．
このことは，翻して考えてみると，要求はしたが実際には使用してない領域（例えば，NULL となっている等の条件が要るが）をマージして実際の消費メモリの容量を小さくすることにおける期待の方が大きい．
実際には，アプリケーション間で同じあるいは類似のデータを扱うことは稀であると予想するが, 仮想計算機側の OS が同じ場合は，プログラムやデータのメモリへの配置は同一になる可能性が極めて高いので，その部分においてメモリが圧縮されることが期待できる．この現象も我々の実験においても確認している．

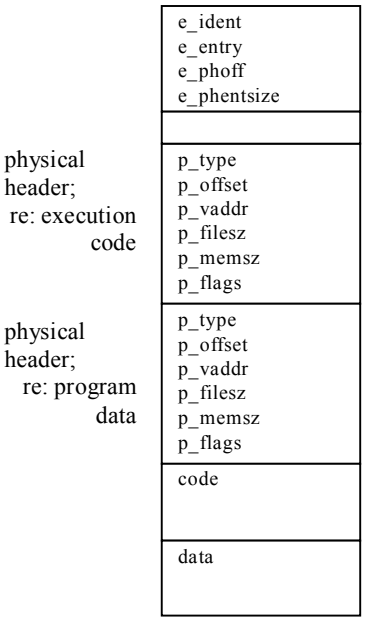
3.2 共有ライブラリー

この KSM の処理の結果として，各々の仮想計算機で使用しているライブラリーが共通のものである場合，同じものとして共有化されることになる，と想定できる．
ライブラリーなどのように同一プログラムをユーザ間で共通的に使用する場合は，例え静的にリンクされていてもマージの対象になれば，実質的に 1 つのものとして扱うことが可能となる，と期待できる[3]．このことにより，互いに独立に稼働している仮想計算機システムではあるが，使用しているライブラリーが同じものであれば，物理的に共有されることとなり，システム全体としてもめもりの消費容量の削減につながる．

4. 共有ライブラリーと ELF

前述の共有ライブラリーは確率的に共有化される可能性がある現象だが，我々は，更に進めて，明示的に仮想計算機の間で共通的に使用する例えばグラフィックスライブラリーのようなモジュールを想定して，意図的にホスト計算機側にこのような共有ライブラリーをユーザ側あるいは仮想計算機側からの要求に応じて設ける手段を検討している．
単一の計算機システムにおいては，Linux のバイナリ―フォーマットである ELF(executable and linkable Format)があり，実行ファイルは完全にメモリに読み込まれる必要はなく，デマンドローディングの手法により，実行イメージの一部がプロセスにより使用される度に，その部分だけがメモリに取り込まれる機構となっている[4].
動的にリンクされた一部は共有ライブラリーとして保持され，実行時にイメージにリンクされる．共有ライブラリーが実行時にイメージにリンクされるときに共有ライブラリーのテーブルはダイナミックリンカーによって使用される．このテーブルには，参照されるすべてのライブラリールーチンに関する情報が含まれている．ライブラリーを探すべき場所と，プログラムのアドレス空間にそれをリンクする

方法を指示する情報である．



Pic.2 ELF Execution File Format

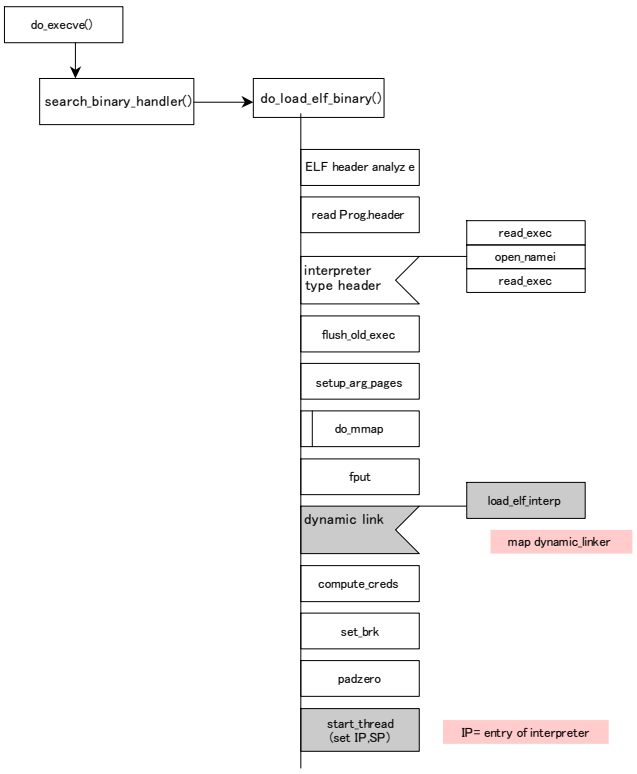
図２は，ELF の実行ファイルフォーマットを示している．実行ファイルには実行コード（テキスト）とデータが含まれる．実行ファイル内のテーブルでは，当該プログラムがプロセッサの仮想メモリに置かれる際の方法を示している．静的にリンクされたイメージは，リンカあるいはリンクエディタによって，そのイメージの実行に必要なすべてのコードとデータを含む単一のイメージにビルドされる．

図２における例では，最初の物理ヘッダは，イメージ内の実行コードに関して，２つ目の物理ヘッダはプログラムのデータに関して記述している．

しかし，ELF 実行イメージをプロセスの仮想アドレス空間にロードするときは，そのイメージをロードするのではなく，仮想メモリのデータ構造，プロセスのツリー(vm_area_struct)，そしてページテーブルのみを設定する．ELF バイナリフォーマットローダは，ロードすべきイメージが有効な ELF 実行イメージを確認し，そのプロセスの現在の実行イメージを仮想メモリから消去し，設定されている全てのシグナルハンドラをクリヤし，オープンしていたすべてのファイルを閉じる．

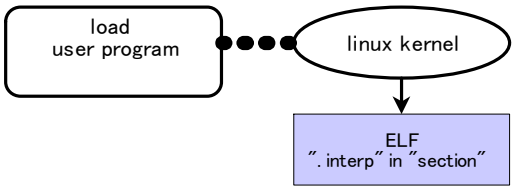
他方，動的にリンクされたイメージは，実行に必要な全てのコードとデータを含まず，共有ライブラリーに保持されていて，実行時にイメージにリンクされる．

ダイナミックリンクが使用される場合，参照されるすべてのライブラリーに関する情報が ELF のイメージテーブルに含まれていて，ダイナミックリンカに対して，ライブラリーを探すべき場所とプログラムのアドレス空間のそれらをリンクする方法を示している．図３に，この一連の処理の流れを示す．



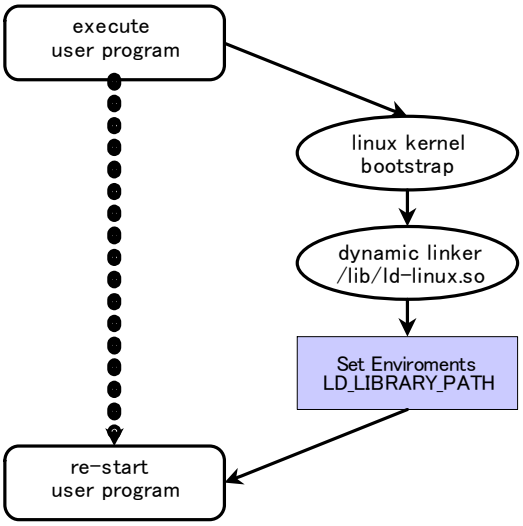
Pic.3 Control Flow of Elf Binary Loader

図４は，ユーザがアプリケーションを起動して ELF イメージが呼出され，カーネルがユーザ空間の仮想メモリにロードするプロセスを開始する状況を示す．カーネルはこのプロセスが動的リンカを使用することを示す ELF セクション”.interp”を検知する．



Pic.4 ELF binary format loader (1)

次に，動的リンクが必要になった際の状況を図５に示す．カーネルは動的リンカをブートストラップし，未だロードされてない指定された共有オブジェクトをロードし，再配置する．そして，使用可能な共有オブジェクトの検索先が LD_LIBRARY_PATH 環境変数により定義される．この処理が終了すると制御が元のプログラムに戻る．このように，単一のコンピュータシステムにおいては，複数ユーザを含む複数プロセス間でオブジェクトを共有することが可能となる機構が提供されている．



Pic.5 ELF binary format loader (2)

5. 仮想計算機上の OS 間共有ライブラリの構築手法

３章にて述べたように, KSM 技術を使用することにより仮想計算機環境においても，消費メモリ容量を低減させることが可能となっている．特に，仮想計算機上で動作する OS が同一のものである場合は，OS 自身が備え持っているオブジェクトが互いに同じあるいは類似であるために，結果として共有化される確率がかなり高くなることが想定されるし，我々の経験でも確認している[1].

我々の研究目標は，既に述べたように，指定したオブジェクトを複数仮想計算機の間でも共有できる機構の実現を目指している．

5.1 Memory map データに注目する方法

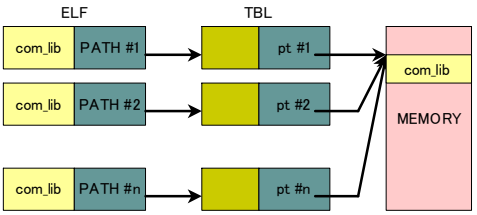
前述のように各 VM における ELF バイナリフォーマットローダが実行されると，最終的にはホスト計算機の匿名領域に関連のデータが収納される．

KSM と同様に, ELF 関連のデータを解析的に探索して各共有オブジェクトが横断的に複数存在しているか否かを探索する．複数存在する場合は，何らかの規則に従って（アドレスの小さい方を基準にする，等）その対象オブジェクトの存在アドレス位置情報に強制的に変更する方法が考えられる．この場合に留意すべきは，当該共有オブジェクトを使用しているユーザ（プロセス）を管理しているデータとの一貫性を保持する必要がある．

例えば，特定の仮想計算機における利用ユーザ数が 0 になっても，他の仮想計算機が当該共有オブジェクトを使用している状態を確認して，すべての利用者がいない状況になってから，当該共有オブジェクトをメモリイメージから削除することが可能となる必要がある．

5.2 環境変数に注目する手法

先ず，LD_LIBRARY_PATH 環境変数に注目して，図６に示すように，対象となる共有オブジェクトの配置先のアドレスが判明すれば，そのアドレス値に仮想計算機間で統一することになる．しかし，このアプローチは，仮想計算機内の振舞を外部からはみることは不可能であるために実現性が小さい．



Pic.6 Forcussing on LD_LIBRARY_PATH in Enviroments

図７に示す mm_struct 構造体には，イメージのコードとデータの始点と終点に対するポインタが存在する，これらの値が分かるのは，ELF 実行イメージの物理ヘッダが読み込まれて，それらのヘッダが記述するプログラムのセクションがプロセスの仮想アドレス空間にマップされたときである．この時同時に，図８に示すプロセスのツリーに関する vm_area_struct が設定され, プロセスのページテーブルが変更されるときである．mm_struct 構造体には，このプログラムに渡されるパラメータと環境変数に対するポインタも含まれている．

```
struct mm_struct {
        struct vm_area_struct * mmap;           /* list of VMAs */
        struct rb_root mm_rb;
        struct vm_area_struct * mmap_cache;
        unsigned long (*get_unmapped_area) (struct file *filp,
              unsigned long addr, unsigned long len,
              unsigned long pgoff, unsigned long flags);
        void (*unmap_area) (struct vm_area_struct *area);
        unsigned long mmap_base;        /* base od mmap area */
        unsigned long free_area_cache;
        pgd_t * pgd;                    /* page graobal directory */
        atomic_t mm_users;
        atomic_t mm_count;
        int map_count;                  /* number of VMAs */
        struct rw_semaphore mmap_sem;       /* semaphore */
        spinlock_t page_table_lock;         /* TBL lock */
        struct list_head mmlist;
        unsigned long start_code, end_code, start_data, end_data;
        unsigned long start_brk, brk, start_stack;
        unsigned long arg_start, arg_end, env_start, env_end;
        unsigned long rss, anon_rss, total_vm,
                                        locked_vm, shared_vm;
        unsigned long exec_vm, stack_vm, reserved_vm,
                                              def_flags, nr_ptes;
        unsigned long saved_auxv[42]; /* ELF interpreter info.*/
        unsigned dumpable:1;
        cpumask_t cpu_vm_mask;              /* CPU mask */
        mm_context_t context;       /* specific MM context */
        unsigned long swap_token_time;
        char recent_pagein;
        int core_waiters;
```

```
        core_startup_done:
        core_done:
        rwlock_t        ioctx_list_lock;
        struct kioctx   *ioctx_list;
        struct kioctx   default_kioctx;
}
```

Pic. 7 mm_struct / linux2.6

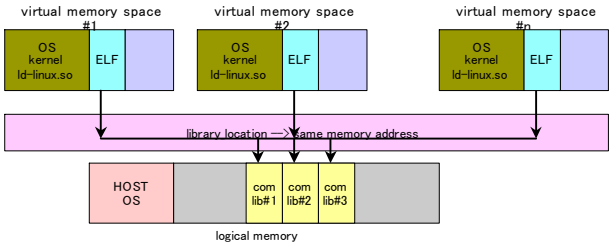
```
struct vm_area_struct {
      struct mm_struct * vm_mm;
      unsigned long vm_start;
      unsigned long vm_end;
      struct vm_area_struct *vm_next;
      pgprot_t vm_page_prot;      /* access permissions */
      unsigned long vm_flags;     /* flag */
      struct rb_node vm_rb;
      union {
        struct {
          struct list_head list;
          void *parent;
          struct vm_area_struct *head;
          } vm_set;
          struct prio_tree_node prio_tree_node;
      } shared;           /* address space link */
      struct list_head anon_vma_node;
      struct anon_vma *anon_vma;
      struct vm_operations_struct * vm_ops;   /* vm operation */
      unsigned long vm_pgoff;
      struct file * vm_file;
      void * vm_private_data;
};
```

Pic.8 vm_area_struct / linux 2.6

5.3 ELF 関連処理に注目する方法

仮想計算機システムにおける複数システムが稼働する形態において，処理を階層構造的に行うことにより，システムの拡張性や柔軟性が一般に期待できる．

このために，各 VM 上の OS のローダが機能する際には，互いの OS システムの間で qemu_kvm を通して[5]，何らかの手段により連携する機構のアルゴリズムを考案する必要がある．図９は，その実現イメージを表している．



Pic.9 Realization image

6. 新 ELF ローダ

本研究では，現在の ELF ローダを本目的に合わせて，改修あるいは新たに製作する方向で進める．

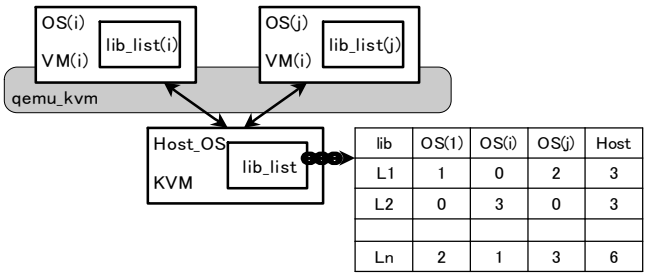
この目標を実現するために，現状の ELF ローダの仕組みをどのように変更するあるいは作り直すことが必要なのかの検討を行う．

6.1 基本的機構

図４に示した環境変数を設定するタイミングで，新 ELF ローダとその関連モジュールが VM 側 OS 間で連携機能する方法と，各 VM 側 OS とホスト OS との間で連携機能する方法が想定される．本研究では，後者の連携方式を対象とする．但し，ELF 関連のデータは，その個別の仮想計算機の環境に依存したものであるので，この環境を超えた制御が必要となる．

6.2 VM 側 OS とホスト OS との間で連携機能する方式

VM(OS)側で ELF ローダが起動された時，あるいは，mm_struct あるいは vm_area_struct が生成される時に，それらの実態をホスト OS 側の管理領域に生成し，各 VM(OS)側から参照・変更できるようにする機構である．この概念を図１０に示す．基本的には，各 VM(OS)内で，（現在のように）共有オブジェクトを使用することを検出してその実行する機構と同じ形態の機構をホスト OS 側に設けて，上記の mm_struct および vm_area_struct に係る各管理データを生成管理する．ホスト側で把握したライブラリーが，それぞれにどの VM(OS)で必要としているものなのかを管理しておく必要がある．なお，これらの情報は qemu_kvm の仕組みを通して検索収集する．



lib	OS(1)	OS(i)	OS(j)	Host
L1	1	0	2	3
L2	0	3	0	3
Ln	2	1	3	6

Pic.10 Collection and management of library data in collaboration with the host OS and the VM OS side

6.3 現 ELF ローダとの関係

仮想計算機内での ELF ローダが起動されたタイミングで，すべてあるいは一部の処理機能はホスト側に移管されて実施される形態とする．処理の実行はホスト側の ELF ローダが行うが，元となる対象ライブラリーの情報は仮想計算機側に存在し，その処理を実際は qemu-kvm の処理機構の中で実施するので，その段階で前記の対象ライブラリーに関する情報を得る事ができると考えている．

6.4 メモリ管理とプロセス管理

この焦点は，共有オブジェクトが配置される物理アドレスの決定である．各仮想計算機の ELF ローダが処理を実施している過程で，ホスト側との連携の中で配置物理アドレス情報を得る，あるいは要求するタイミングがある．この交信処理を新たに構築する必要がある．

関連する課題は，仮想計算機の間で共有するオブジェクトの判定である．個別仮想計算機内で，共有するあるいは共有する可能性がある場合は，ホスト側から見ても唯一の仮

想計算機内でのみ共有が行われていても，ホスト側で「共有」対象として扱う．
他方，個別計算機内では，共有するあるいは共有する可能性が無い場合でも，ホスト側から見て他の仮想計算機の間で共有するケースをどのようにして検出するかに関する処理機構が必要になる．
以上の事柄は，プロセスの生起に強く関連している．

6.5 Qemu_kvm との関係

未だ本 qemu_kvm の解析は不十分であるが，複数コア CPU を前提とした構成になっている．仮想計算機の資源割当てにおいても複数 CPU の使用が可能であり，個々の CPU に対応したメモリなどの資源対応の管理機構が心材する．関連して例えば，処理途中におけるプロセスの移送なども行われる際には，メモリ管理テーブル TLB と実メモリ配置の関係も追随して一貫性を保ちながら処理が行われる．このような事象においても，対象となる共有オブジェクトの存在するメモリ位置情報の一貫した管理にも対応できるようにすることも必須条件となる．

7. おわりに

KSM の機構を利用することにより，メモリ内のデータをページ単位で比較することにより，同一データが存在するページ領域を，１つに圧縮してそれを共有して利用することによりメモリの消費量を削減できる[12]．この処理による圧縮量の結果は確率的なものとなる．
そこで，我々は，KSM を用いた場合に仮想計算機環境における使用メモリ容量をコミットするための指針を得るべく評価実験を行った経験を基に，予め共有する，あるいは共有の可能性のあるオブジェクトを意識した上で，仮想計算機環境全体で構造的に共有して利用する機構を検討した．特に，同一のあるいは類似の OS が備えている共通のライブラリーを，KSM の機構とは別に，メモリ管理機構において管理している当該共通オブジェクト（ライブラリー）の格納されている場所データを，ホスト OS が司るメモリ管理機構が管理する共通領域に反映させて re-entrant な共通ライブラリーとして管理する手法を想定している．最近の Linux に導入されている LD_LIBRARY_PATH 環境変数と関連する管理データ(プログラムの配置に関する mm_struct, プロセスの配置に関する vm_area_struct，等)に注目して，仮想計算機とホスト間で連携する方式を目指している．これらに基づく構築手法の検討状況を述べた．
この手法により，メモリの消費容量を削減できる可能性はあるが，これは我々の研究の最終目的ではない．
グリッドコンピュータシステム，あるいはクラスターコンピュータシステムを対象にした，共有メモリの手法によるオブジェクトの共有や通信方式の研究はみられるが[7], OS 間で連携する機構を目指した共有連携の研究は少ないようである[8][10][11][13]．構成形態が何であれコンピュータシステムが互いに広い領域で連携した環境を提供できるようになることにより[6]，本来的な意味での情報処理基盤環境としてのインフラが構築できると考える．

参考文献

1)中尾司ピエール，坂下善彦：Linux 仮想環境におけるメモリコミットの分析，情報処理学会 DPS 研究会報告，2012.9.14
2)Andrea Areangeli, Izik Eidus, and Chris Wright: Increasing memory density by using KSM, Proceedings of the Linux Symposium(2009)
3)M. Tim Jones:v Anatomy of Linux Kernel Shared Memory, <http://www.ibm.com/developerworks/linux/library/l-kernel-shared-memory/ >(2013.01.11)
4)Abraham Silberschatz, Peter Baer Galvin, Greg Gagne: Operating System Concepts, John Wiley & Sons, Inc. 2005
5)Daniel Bartholomew: QEMU: a multihost, multitarget emulator, Linux Journal, Volume 2006 Issue 145, May 2006
6)Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Lsaacs, Simon Peter, Timothy Roscoe, Adrian Schupbach, Akhilesh Singhania: The Multikernel: A New OS Architecture for Scalable Multicore Systems, SOSP'09, Oct. 2009
7)Kolhe, S. ; Dept. of Comput. Eng., Mahatma Gandhi Mission, Navi Mumbai, India ; Dhage, S.: Comparative study on Virtual Machine Monitors for cloud, Information and Communication Technologies (WICT), pp. 425 – 430, 2012 World Congress on Oct. 30 2012-Nov. 2 2012
8)Chao-Rui Chang ; Acad. Sinica Dept. Comput. Sci. & Inf. Eng., Nat. Taiwan Univ., Taipei, Taiwan ; Jan-Jan Wu ; Pangfeng Liu: An Empirical Study on Memory Sharing of Virtual Machines for Server Consolidation, Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium on, 26-28 May 2011, pp. 26-28 May 2011
9)Wei Dong ; Zhejiang Key Lab. of Service Robot, Zhejiang Univ., Hangzhou, China ; Chun Chen ; Xue Liu ; Jiajun Bu: Dynamic linking and loading in networked embedded systems, Mobile Adhoc and Sensor Systems, 2009. MASS '09. IEEE 6th International Conference on, 12-15 Oct. 2009, pp554 – 562
10) Chao-Rui Chang ,Jan-Jan Wu, Pangfeng Liu,: An Empirical Study on Memory Sharing of Virtual Machines for Server Consolidation,
11) ISPA '11 Proceedings of the 2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications, pp.244-249
12) Prateek Sharma, Purushottam Kulkarni: Singleton: system-wide page deduplication in virtual environments, HPDC '12 Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing,pp.15-26
13) Timothy Merrifield, Jakob Eriksson: CONVERSION* Multi-Version Concurrency Control for Main Memory Segments, Eurosts'13, pp.127-139,April 2013
14) Umesh Deshpande, Xiaoshuang Wang, Xiaoshuang Wang: Live gang migration of virtual machines, HPDC'11 Proceedings of the 20th international symposium on High performance distributed computing , pp.135-146, 2011