目次

第一章　クラウドと仮想マシン
    （ア）クラウドとデータセンタ
    （イ）仮想マシン
    （ウ）Google Compute Engine
第二章　ライブマイグレーションの理論
    （ア）ライブマイグレーションとその利用
    （イ）Pre-copy ライブマイグレーション
    （ウ）Post-copy ライブマイグレーション
    （エ）まとめ
第三章　ライブマイグレーションの実際
    （ア）QEMUにおけるライブマイグレーションの実装
    （イ）Pre-copyにおけるメモリ更新検知の実装
    （ウ）Post-copyにおけるオンデマンド転送の実装

第一章　クラウドとGoogle Compute Engine

（ア）クラウドとデータセンタ

クラウド、あるいはクラウドコンピューティングという語は現在では日常的に聞かれるようになりました。**クラウドコンピューティングとは、インターネットの雲（Cloud）のむこうから計算リソースを必要な時に借りる形態**を言います。(最近よく聞く似た言葉に「クラウドソーシング」がありますが、こちらは大衆（Crowd）の持つお金や労働力などのリソースを少しずつ集めて大きな成果を得ようというもので、クラウドコンピューティグとは直接は関係ありません。)

さて、インターネットの雲の向こうにあるクラウドの裏では、「データセンタ」と呼ばれる施設にコンピュータが何千台も並べられて動作しています。データセンタで使われるコンピュータは中身は家庭のものとほとんど同じですが（一方、「京」や「Cray」などのスーパーコンピュータは専用の通信機構を持っている等特殊な設計になっています)、**一台では家庭にもあるようなコンピュータでも、何千台も並べると電力・冷却・故障など様々な問題が発生**します。例えば冷却ではデータセンタの消費電力のうち半分近くがエアコンに使われてしまうという問題が発生し、それを避けるために北海道や東北などの寒冷地にデータセンタを作る会社も存在します。

（イ）仮想マシン

ユーザがクラウドから借りてくる計算リソースには様々な種類がありますが、Google Compute EngineやAmazon AWS で提供されている形態のうち最もポピュラーなものが仮想マシン（Virtual Machine; VM）です。**仮想マシンとはあるコンピュータの上で仮想的に別のコンピュータを動かす技術**です。身近な例ではMac 上のVMWareやParallelsにWindowsをインストールしたり、Windows 8上のVirtualPCでWindowsXPを動かすことがあります。このとき、Macのハードウェアを物理マシン、Mac OSをホストOS、VMWare等を仮想マシン、WindowsをゲストOSと呼びます。

クラウドではユーザは仮想マシンを借り、その上に様々なサービスを展開します。クラウドで提供する計算リソースとして仮想マシンを利用する

ことはユーザ側、事業者側の双方にとって利益があります。

ユーザ側：借りた仮想マシンを占有できるため、ほかのユーザを気にせず好きなソフトウェアをインストールしたり負荷をかけたりできる。

事業者側：仮想マシンは物理マシン（実際のコンピュータ）の上にソフトウェアで作られているため、本稿で扱うライブマイグレーションのように物理マシンではできない様々な管理が可能である。ライブマイグレーションによってどんなことが可能になるかは第二章（ア）で説明します。

（ウ）Google Compute Engine

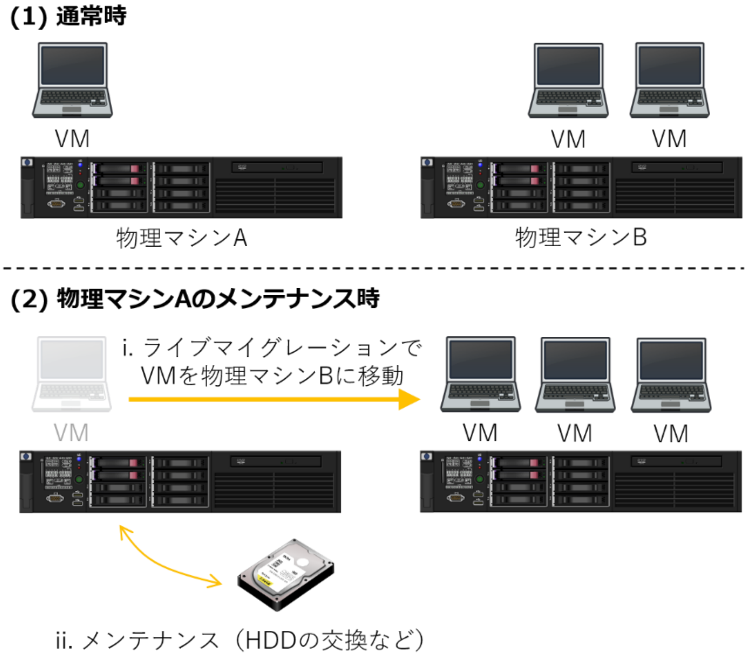
Google Compute Engineは、数あるクラウドサービスのうちGoogleが提供するものです。クラウドサービスにはほかにもAmazon AWS（Amazon）、さくらのクラウド（さくらインターネット）、ConoHa（GMOインターネット）、Cloud^n（NTTコミュニケーションズ）など様々なものがあります。本書で特にGoogle Compute Engineをと書いているのは、ライブマイグレーションをデータセンタで実際に使っているというプレスリリースが話題になったからで（実際には他の事業者も発表しないだけで使っているかもしれません）、本書の内容が特にGoogle特有の話というわけではありません。

第二章　ライブマイグレーションの理論

（ア）ライブマイグレーションとその利用

仮想マシンのライブマイグレーション（Live Migration）とは、仮想マシンを停止させずにある物理マシンから他の物理マシンへ移動させる技術です。この技術は2005年ごろによって提案されました（逆に言えばライブマイグレーションは「Googleによって提案された最新技術」というわけではありません）。ライブマイグレーションは、仮想マシンを中で動いているプログラムや通信の状態を壊さないまま丸ごと移動することができ、仮想マシンのユーザからみると何も起きていないように見えます。

仮想マシンのライブマイグレーションはクラウド上で様々な使いみちがあります。その最たるものが、物理マシンのメンテナンスです。クラウド（データセンタ）ではサーバが何千台と動いているため毎日どこかでメンテナンスや故障が起こっています。一方ユーザと契約している SLA（Service Level Agreement; これくらいのサービスレベルを維持しますよ、という契約）では、契約期間のうち 99.99%の期間停止せずに使えるなどとなっています。この契約では例えば 1 週間のうち 40 分しか停止時間を確保できませんので、**物理マシンのメンテナンスをその上で動く仮想マシンを停止させずに行う**必要があります。そこで、**メンテナンス対象の物理マシンの上で動いている仮想マシンを別の物理マシンにライブマイグレーションし**、空いた物理マシンをメンテナンスすることでこれを実現できます（下図)。



Google のクラウドである Google Compute Engine では、ライブマイグレーションをまさにメンテナンスのために利用しています。2015 年 3 月 25

日の Google によるブログ（http://googlecloudplatform-japan.blogspot.jp/2015/03/google-compute-engine.html）では以下のように発表されています。

「Google Compute Engine にトランスペアレント メンテナンス（透過的なメンテナンス）を導入しました。… データセンタートポロジーのイノベーションと、ライブマイグレーションテクノロジーを組み合わせることで、定期的なハードウェアやソフトウェアのメンテナンスがあったとしても、それに VM が影響を受けないようにし …」

（イ） Pre-copy ライブマイグレーション

ライブマイグレーション技術の概要と使われ方に続いて、実際に仮想マシンをライブマイグレーションするにはどうすればよいか見ていきます。まず、最も原始的な方法である「Pre-copy ライブマイグレーション」を解説します（Google のブログで解説されているものと同じですが、もう少し詳しく解説します）。

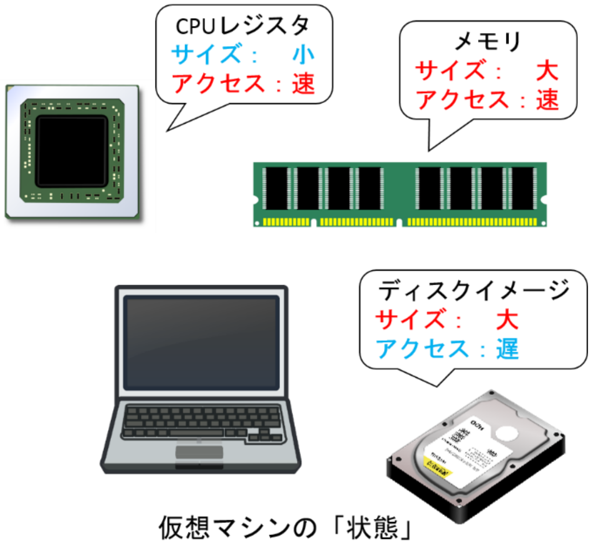
仮想マシンをライブマイグレーションするには、仮想マシンの「状態」を全て移動先の物理マシンに転送する必要があります。仮想マシンの「状態」は、次の 3 つに大別できます。

1. 仮想マシンのメモリ内のデータ
2. 仮想マシンのディスクイメージ
3. 仮想マシンの CPU やデバイスのレジスタなどその他の状態

このうち、ライブマイグレーションで問題になるのは 1 のメモリ内のデータです。ディスクイメージは一般に NFS などの分散ファイルシステム上に設置し、すなわち移動元と移動先の物理マシンから同じ状態が見えるようにするため転送する必要がありません。また 3 の CPU レジスタ等はサイズがとても小さいため簡単に転送できます。一方、メモリはアクセスが速いくかつサイズが大きいため、次のような課題が発生します。

1. ディスクのように分散環境に置くと遅延によって仮想マシンの性能が下がる → **ライブマイグレーションを行う際に転送する必要**

2. レジスタ等に比べてサイズが大きい → **転送するあいだ仮想マシンを停止させると仮想マシンが長い時間止まってしまう**



仮想マシンの「状態」

さて、ライブマイグレーションの目的は 99.99%等の SLA を守るため仮想マシンを止めずに物理マシンのメンテナンス等を行うことでした。従って 2 の「仮想マシンが長い時間止まってしまう」は許容できません。そこでこの課題を解決するため、Pre-copy 型のライブマイグレーションでは**仮想マシンを動かしたままメモリを転送**します。

仮想マシンを動かしたままメモリを転送すると何が起きるでしょうか。**仮想マシンが動いているということはメモリのデータが更新されるので、一度転送したデータをもう一度転送する**必要が生じます。従って、Pre-copy 型ライブマイグレーションにおけるライブマイグレーションの手順は以下のようになります。

1. 仮想マシンのメモリを先頭から順に移動元の物理マシンから移動先の物理マシンに転送する。
2. 仮想マシンが動いていることによって更新されてしまったメモリをもう一度転送する。転送すべき残りメモリ量が十分少なくなるまでこれを繰り返す。

3. 転送すべきメモリ量が十分少なく（例：数MB）になったら、仮想マシンの実行を一瞬だけ止めてCPUレジスタ等を転送する。

仮想マシンを停止させるのは3の残り少なくなったメモリとCPUレジスタ等を転送する間だけですから、仮想マシンが長い時間止まってしまうことはありません。

（ウ）Post-copyライブマイグレーション

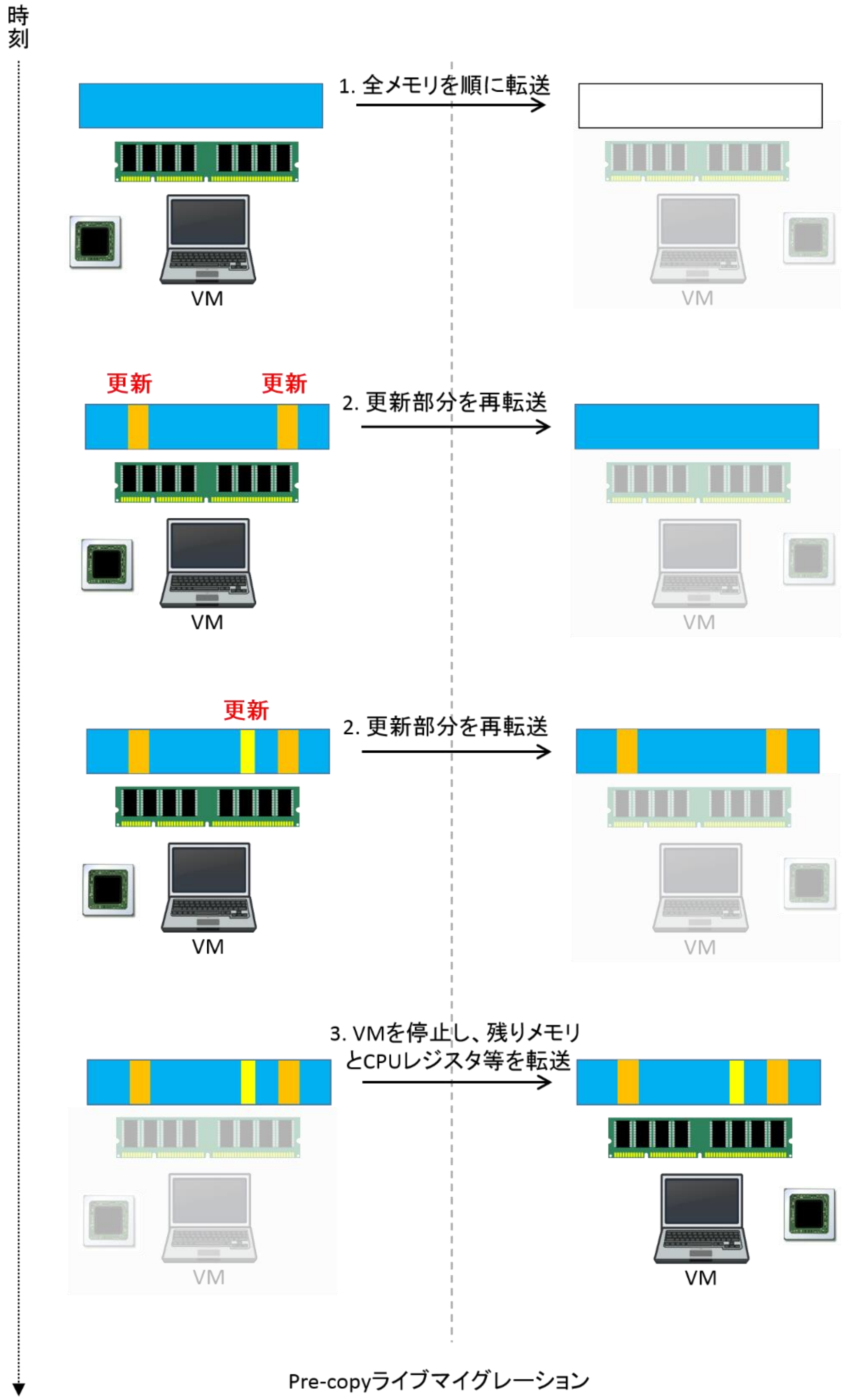
（イ）で説明したPre-copyライブマイグレーションでは、更新されたメモリの再転送によって転送すべき残りメモリ量が十分小さくなる必要がありました。しかし、仮想マシンのメモリ更新が速く転送が追いつかない場合にはこれは不可能で、永遠に2の再転送を繰り返してしまいます。

そこで考案されたのが、Post-copy型のライブマイグレーションです。Post-copy型では、**メモリの内容をCPUレジスタ等の後に転送する（post-copy）によってメモリの再転送を不要に**します。すなわち、Post-copyライブマイグレーションの手順は以下のようになります。

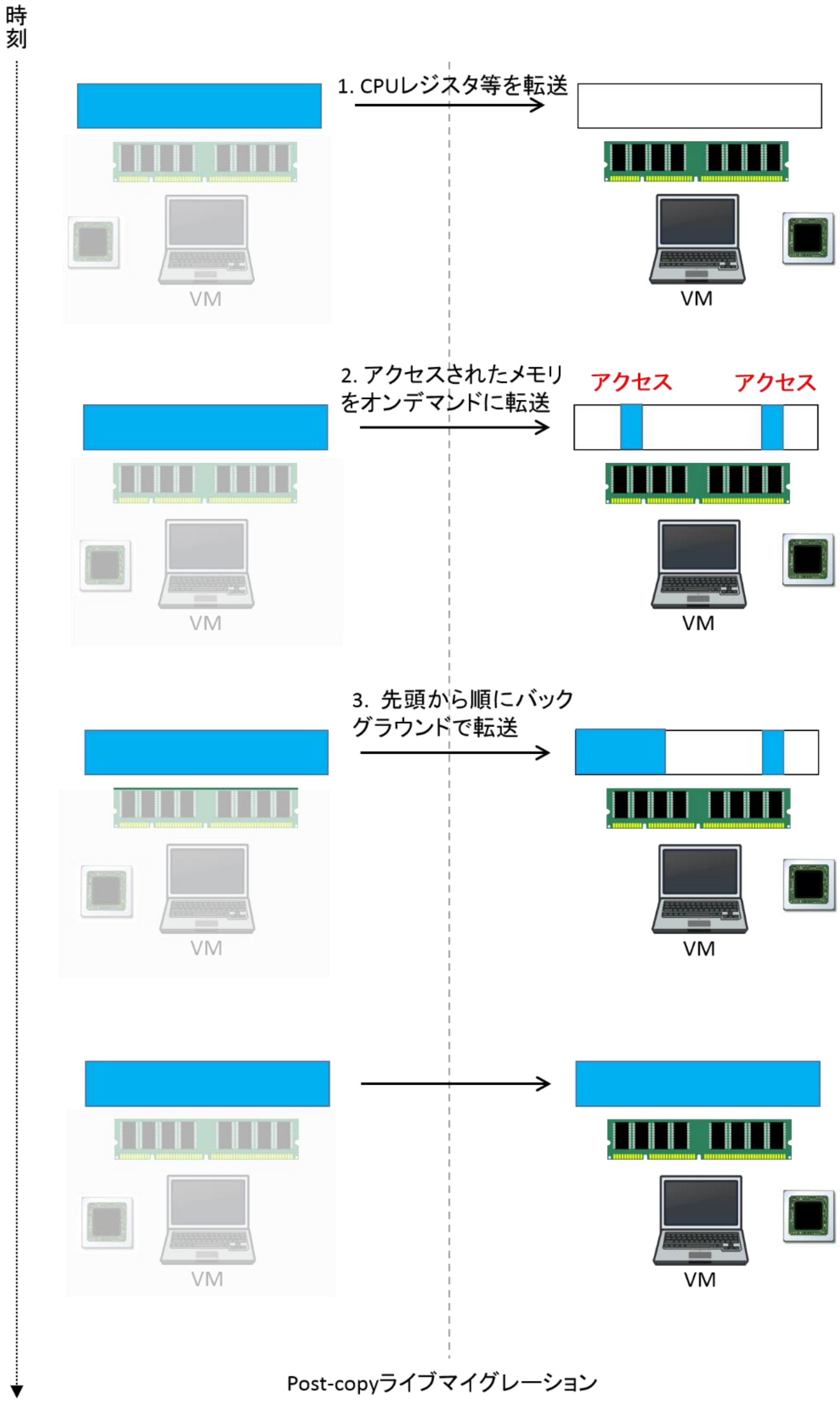
1. 仮想マシンのCPUレジスタ等を転送する。**この時点で仮想マシンの実行は移動先の物理マシンに移る。**
2. 移動先で実行されている仮想マシンでアクセスのあったメモリをオンデマンドに転送する。
3. 移動元の物理マシンに残っているメモリ内容を、2のバックグラウンドで先頭から順に転送する。

この方式のメリットは、**どんなに仮想マシンのメモリ更新が速くてもマイグレーション可能であること**です。移動元にある仮想マシンの状態が1の時点から更新されずPre-copyライブマイグレーションで課題だった更新されたメモリの再転送が発生しないためです。逆にこの方式のデメリットは、**全メモリの転送が終了するまで仮想マシンの性能が下がってしまう**ことです。2でアクセスされたメモリをオンデマンドに転送する際、仮想

マシンはメモリが転送されてくるまで待たなければなりません。その間当該メモリにアクセスしたアプリケーションは実行を停止しなくてはいけないので、性能が下がってしまいます（やや難しい説明：ただし実際にはメモリを要求したプロセスが待っている間に別のプロセスがスケジューリングされるため、メモリが転送される間丸々仮想マシンが止まるわけではありません）。



Pre-copyライブマイグレーション



Post-copyライブマイグレーション

（エ）まとめ

仮想マシンのライブマイグレーションは仮想マシンを動かしたまま複数の物理マシン間で移動する技術で、物理マシンのメンテナンスの影響を最小限にするために実際にGoogleのクラウドで使われています。

ライブマイグレーションの方式にはPre-copyとPost-copyの二種類があり、それぞれの特徴をまとめると以下のようになります。

	Pre-copy	Post-copy
メモリの転送	CPUの前（pre）	CPUの後（post）
特徴	更新されたメモリは再転送。メモリの更新が速いと適用できない。	メモリの更新が速くても適用可能。仮想マシンの性能がしばらく下がる。

第三章　ライブマイグレーションの実際

（ア）QEMUにおけるライブマイグレーションの実装

この章ではQEMUと呼ばれる仮想マシンモニタ（仮想マシンを実現・管理するソフトウェア）のソースコードを使って実際にライブマイグレーションがどのように実現されているのか見ていきます。

QEMUのソースコード（2015年12月28日時点の最新版は2.5.0）は下記からダウンロードすることができます。

http://wiki.qemu-project.org/download/qemu-2.5.0.tar.bz2

ダウンロードしたものを展開すると多くのファイルやディレクトリがありますが、マイグレーション関係のものは migration ディレクトリの下に全て入っています。

（イ）Pre-copyにおけるメモリ更新検知の実装

Pre-copyライブマイグレーションを実装する上で重要な機能は、ある時

点からスタートしてメモリの更新された部分を知ることです。ただし、例えばメモリのコピーを丸々もっておいていちいちどこが変わったか比較・・・とはできません。メモリは数GBから数十GBあり、こんなことをしていては更新部分を知るのに何秒、何十秒もかかってしまいます。

そこでQEMUでは、**メモリの更新を検知するのに dirty flag を使います**。dirty flag とは、メモリのある部分に書き込みがあった際に自動的に 1 になるフラグで自由に読み込みおよびリセットすることができます。QEMUではこのフラグを用いて、次のようにメモリ更新の検知を実現します。

1. メモリのある部分を転送するたびにその部分の dirty flag を 0 にクリアする。
2. メモリの転送が一巡した後、あるメモリ部分の dirty flag が 1 になっていればそのメモリが更新されたことを意味する。従って当該メモリ部分を再転送する。

やや難しい説明：dirty flag は実際には CPU が提供する機能で、Intel の“Intel® 64 and IA-32 Architectures Software Developer’s Manual Volume 3A: System Programming Guide, Part 1”の“4.8 ACCESSED AND DIRTY FLAGS”に次のように記されています（翻訳は筆者に因る)。

「… あるリニアアドレスに対し書き込みが起こったとき、プロセッサは（既にセットされていなければ）そのリニアアドレスが指す最終段の物理アドレスに対応するページ構造体に dirty flag をセットする ***（訳注：たぶんちゃんとメモリアドレス変換をして一番下の物理アドレスのみ扱いますよ、という意味)。*** *… メモリ管理ソフトウェアはあるページ* ***（訳注：メモリページ）*** *またはページ構造体がメモリに読み込まれた時にこれらのフラグをクリアできる。これらのフラグは”sticky”である、つまり、一度セットされればプロセッサによってはリセットされずソフトウェアのみがリセットできる。」*

さて、dirty flag によるメモリ更新の検知は QEMU のソースコードでは migration/ram.c の ram_save_iterate 関数（1962行目）に以下のよ

うに書かれています。関数名が save_iterate でイテレーティブに（何度も）メモリを転送する様子を現しています。

```
int pages;

pages = ram_find_and_save_block(f, false, &bytes_transferred);
/* no more pages to sent */
if (pages == 0) {
    break;
}
pages_sent += pages;
```

ram_find_and_save_block でどのメモリ部分が更新されたかを取得し、もしそれがゼロ（つまりもう送るべきページがない “no more pages to sent”）なら処理を終了します。さらに ram_find_and_save_block は 1323 行目から定義されており、追いかけていくと結局次の二つの関数にたどり着きます。

551 行目：`ram_addr_t migration_bitmap_find_dirty(...)`
574 行目：`bool migration_bitmap_clear_dirty(...)`

これ以上もぐるのは複雑になってくるので記せませんが、関数名から dirty flag のビットマップを見つけたりクリアしたりしているのが分かると思います。

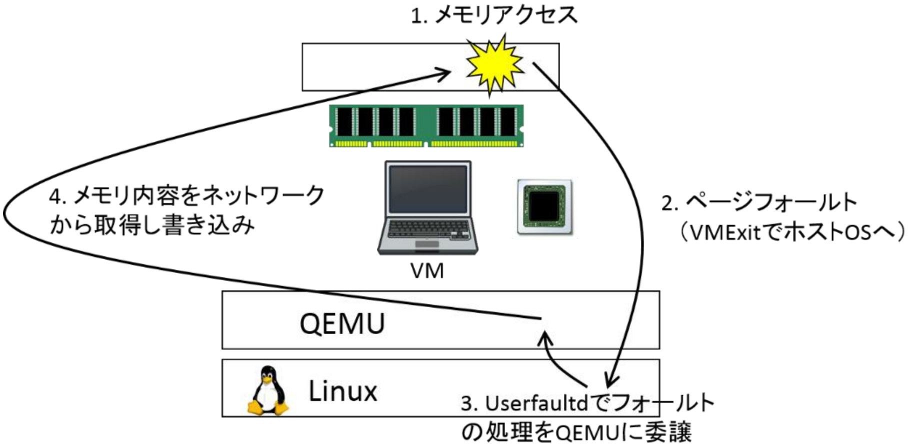
（ウ）Post-copy におけるオンデマンド転送の実装

Post-copy ライブマイグレーションでは、Pre-copy にはない移動先でアクセスされたメモリをオンデマンドに移動元から転送する処理があります。すなわち、**仮想マシンによるメモリのアクセスをその都度検知して捕まえる**必要があります。

QEMU ではこの仕組みを**ページフォールトを検知して補足する**ことによって実現します。ページフォールトとは、あるメモリアドレスにアクセスしたがそのデータがメモリ中に存在しなかった場合に発生する例外で、通

常はメモリのスワップ（メモリが足りないときに HDD に一時的にデータを退避する仕組み）で利用されます。

やや難しい説明：Linux ではページフォールトを OS 以外のソフトウェアが効率的に扱うため仕組みである Userfaultd を提供しています。Userfaultd ではあるメモリアドレスの範囲に対してフックするページフォールトの種類を登録することで、指定された範囲に対して指定されたフォールトが起こった際にファイルディスクリプタ経由で通知してくれます。Userfaultd の詳しい説明と、それを用いた Post-copy ライブマイグレーションの実装方法の説明が Linux の公式ドキュメント内（https://github.com/torvalds/linux/blob/master/Documentation/vm/userfaultfd.txt）にあります。



Post-copy ライブマイグレーションの実装は、QEMU では migration/postcopy-ram.c の中にあります。具体的には 375 行目からの ram_block_enable_nity 関数で Userfaultd による通知を受けるよう登録し、 398 行目からの postcopy_ram_fault_thread 関数 Userfaultd によるページフォールトの通知を待って通知が来たらメモリ転送処理をしています（日本語によるコメントは筆者による注釈）。

```
static int ram_block_enable_notify(…)
{
    ...
    if (ioctl(mis->userfault_fd, UFFDIO_REGISTER, &reg_struct)) {
        /* Userfaultd に登録し、失敗すればエラー */
        ...
        return -1;
    }

    /* 成功 */
    return 0;
}
```

```
static void *postcopy_ram_fault_thread(void *opaque)
{
    ...
    /* Userfaultd からの通知を受け付け続ける */
    while (true) {
        ...

        /* enable_notify で登録したディスクリプタから読み出し */
        ret = read(mis->userfault_fd, &msg, sizeof(msg));

        /* エラー処理  */
        ...

        /* ページフォールトが起こったアドレスを移動元に通知し、
          当該メモリの内容を送ってもらうように要求 */
        rb = qemu_ram_block_from_host(
                    (void *)(uintptr_t)msg.arg.pagefault.address,
                    true, &in_raspace, &rb_offset);

    ...
}
```

・本書の図は openclipart（https://openclipart.org/)より引用しました。
・本書のカラー版 pdf を http://www.soramichi.jp/pdf/C89.pdf に用意していますので是非ご利用ください。アクセス制限/配布制限等ありません。
・本書はコミックマーケット８９において無料頒布したものです（委託先：1 日目東キ 11b あいすまぐねっと 3 日目東ム 37b 縁の下ねっとわーくす）。