QEMU-KVM虚机动态迁移原理

在虚拟化领域，虚机动态迁移是一个非常有趣且持续不断的话题：因为随着使用需求，客户机变得越来越大（单个虚机的vCPU和RAM越来越多）且客户机正常运行不间断的需求也变得越来越严格。

本次讨论将围绕QEMU/KVM hypervisor早期在线迁移的简单设计，例如到它是如何优化到当前的技术能力，未来将如何发展。这里会探讨在线迁移真正的工作原理，限制条件以及如何开发新思路设计来满足最新在线迁移需求。

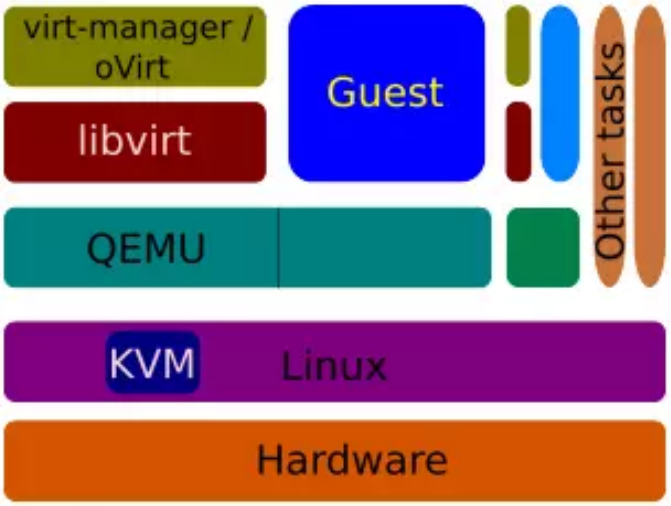
本次讨论将涉及到已知的，未知的技术领域（例如TODO）以供感兴趣的研究者再接再厉。

本文源于我最近发表在the devconf.cz conference.上的一个演讲。胶片的PDF版可供参考，但你并不需要它来理解本文。本文是胶片的文本表示，是内容的详细描述，对应我在演讲过程中的阐述。

目录

- Subsection Example
- Things　Changed Recently 最新变化技术
- New　Features 新特征
- Stuff　That’ s Lined Up
- Future Work

Virtualization虚拟化



我们从虚拟化栈总图的左上角开始，用户与虚机交互通过几个界面：例如virt-manager, oVirt, OpenStack, or Boxes.这些软件依次与Libvirt做交互，Libvirt可以提供支持多个hypervisor来管理虚机的API。Libvirt还提供与多种块存储和网络设置做交互的API接口。

对于QEMU/KVM虚机，Libvirt使用QEMU API与QEMU对话，宿主机上的每台虚机都是QEMU实例，客户机作为QEMU进程的一部分运行。使用top或者ps命令，可以看到每一个vCPU就是宿主机中单独的一个线程。

QEMU与Linux做交互，尤其是Linux内部KVM模块交互，是基于物理硬件运行，并不是在QEMU模拟环境中。

QEMU

- Creates the machine 创建虚机
- Device emulation code 模拟设备

    - some mimic real devices 模拟类型真设备（纯软虚拟化设备）
    - some are special: paravirtualized 半虚拟化设备
- Uses several services from host kernel 源宿主机内核设备
    - KVM for guest control 客户机控制器KVM
    - Networking 网络设置
    - disk IO 磁盘IO
    - etc.

QEMU是一个虚拟化开源软件，创建虚拟硬件环境来运行客户机的操作系统。所有的虚拟硬件都是基于QEMU配置代码创建的。客户机操作系统，例如键盘，鼠标，网卡等等设备均是QEMU代码实例，这些虚拟设备是基于技术参数的，开放给物理硬件使用，例如e1000系列的网络设备。由于这些物理硬件的驱动已经经过调配，不需要经过修改的客户机操作系统都可以识别这些设备。

除了模拟真实硬件（全虚拟化硬件），QEMU也专门为提升虚拟化使用性能创建一些特殊的设备。对于QEMU/KVM虚机，我们使用virtio架构来创建这些设备，包括virtio-net网络设备，virtio-blk块设备，virtio-scsi SCSI设备，virtio-rng RNG设备等等。这些半虚拟化设备得益于采用虚拟化思路设计，所以它们比全虚拟化设备运行更快，更容易管理。

QEMU也与其他程序做交互，例如为客户机提供BIOS服务的SeaBIOS。
在操作上，QEMU借助源宿主机Linux 内核的几个设备，例如使用KVM APIs来控制客户机，还会使用源宿主机网络设置和存储设备等等。

KVM

- Do one thing, do it right做一件事就要将它做好
- Linux kernel module Linux内核模块
- Exposes hardware features for virtualization to userspace将硬件支撑虚拟化特征呈现给用户
- Enables several features needed by QEMU提供QEMU所需特征
- like keeping track of pages guest changes记录客户机变化页

KVM是一个小型内核模块，可以给Linux内核提供硬件虚拟化特征。这种代码负责将Linux内核转化成hypervisor，KVM以典型的Linux或者Unix形式写出来，它把内存管理，进程调度等工作交给Linux内核来决定这意味着，虚拟化层将及时受益于任何有关Linux内存管理的性能提升。

KVM将其API通过ioctls这个控制套接口暴露给用户空间，QEMU是这些设备的用户之一。KVM暴露给用户空间的特征之一是追踪客户机内存区域页，例如前一个时段数据申请后客户机进行修改等动作。这个特征尤为重要，我们在整个在线迁移过程都会使用到它。稍后我们会更多地谈论到这点。

LiveMigration在线迁移

在客户机运行过程中，从一个QEMU程序中获取客户机状态，将其迁移到另外一个QEMU程序中。
客户机不会感知到整体环境的变化。换句话说，客户机不会牵扯到这个过程中，尽管如此，也许会感知到一点性能的降低。

- 对负载均衡，硬件和软件的维护，省电，检测等都有用处。

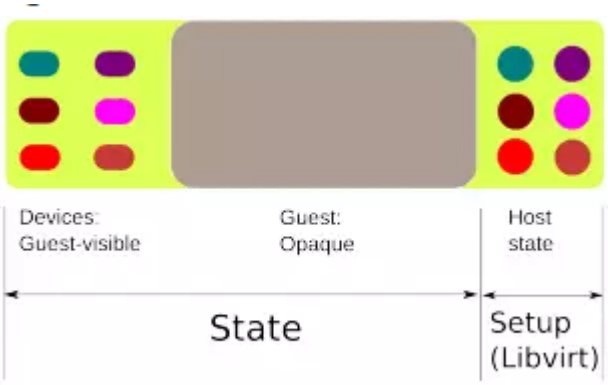
在线迁移就是在客户机仍在运行的情况下，将一个客户机VM从一个hypervisor/QEMU程序，在另外一个上运行起来。客户机持续正常工作，甚至不会意识到hypervisor的变化。在线迁移是将整个客户机从一个物理宿主机转移到另外一个。这意味着会涉及到很多变量：网络带宽，网络延迟，存储可用性等等。

使客户机停止运行并让新的hypervisor接管从而运行该客户机的窗口期非常短。这样可能导致客户机应用程序性能降低，这是唯一使客户机在迁移过程中暴漏之处。

在线迁移由很多好处：在一个服务器池中的几个主机之间做负载，或者在客户机运行的情况下，关闭一些主机对其硬件或者软件进行维护，或者省电（如果一些主机主机在一个集群中是轻负载的，并以可以承载更多VM而不影响服务，那么更多的VM可以迁移到这个轻负载服务器上，关闭那些没有使用的VM。

工程师还可以检测VM，重新运行特定的流量，或者负载在VM上重复检测，或者在VM上进行软件调试。尽管检测这种用户案例与再现迁移无关，但是过去常常设置检查点的代码在QEMU内部是与在线迁移代码共享的。

QEMU布局



让我们看看在迁移过程中涉及到的一些东西。我们需要迁移一些客户机运行的现状，也就是不用翻译任何关于内存区域的内容--整个客户机。这个客户机被迁移代码当做一个模糊的东西，我们只是将这些内容送到目的地，这个区域在以上图表中被标记为灰色。

图表中最左边的部分，就是客户机可见的设备状态，通过QEMU发送协议的状态。包括每一个暴露给客户机的设备状态。

最右边部分的图表是QEMU状态，整个状态不涉及迁移过程。但是在迁移开始之前将这个状态设置正确很重要。在源主机和目的主机上的QEMU设置必须相同，这是通过在两个主机上使用一条相似的QEMU命令行来实现的。由于QEMU命令行的多种选择非常难以搞定，我们使用Libvirt替我们选择正确的。Libvirt可以确保QEMU两边程序的迁移设置正确性。对于Red Hat Enterprise Linux，运行QEMU不是首选，甚至在一些新版本上都不支持，所以我们都是通过Libvirt来使用QEMU的。

我们将会在接下来的环节，介绍有关这三个部分的更详细的内容。

迁移的正确配置

- Shared storage 共享存储
  - NFS settings NFS设置
- Host time sync主机时间同步

    - Can’ t stress enough how important this is! 这点非常重要
- Network configuration 网络设置
- Host CPU types 主机CPU类型
- Guest machine types客户机类型
    - esp. if migrating across QEMU versions 如果迁移要夸QEMU版本需要主机客户机类型
    - ROM sizes ROM大小

从图标的最右边开始看，将源主机和目标主机的设置相同是迁移的成功的关键。看起来将两台VM设置相同很简单，但实际上有些细节并不那么显而易见。

VM使用的存储必须共享，对于源主机和目标主机都是可用的。同时，共享是如何实现的也很重要，例如，文件系统使用必须被设置成取消缓存功能（否则来自源主机的动态数据，在目标主机开始运行并访问存储前，可能不会读写到存储中，导致客户机看到损坏的数据。）

两端服务器的时钟时间必须匹配一致。这里可以使用NTP协议。但是设置正确的时区以及确保两端主机的时间是相同的，这两点也是至关重要的。我们在迁移初始成功的时候，已经有了几个错误报告，而后期客户机会变得迟钝，或者完全停止运行。这个时候，很难在主机的时间间隔解决掉这个问题。对于这种错误设置，客户机会毫无抵抗能力。

两端主机的网络设置必须匹配一致。如果客户机之前在和其他服务器通话，那么对那些服务器的访问要在迁移后进行。防火墙设置在这里也同样重要。同时，在迁移过程中最好将存储网络和网络分开。带宽可用的情况下迁移会快很多，所以存储IO读写最好不占用网络。

主机CPU类型必须一致。暴露给源主机客户机的指令组必须对目标主机客户机完全可用。这意味着从新一代处理器迁移到旧一代处理器可能不成功，除非非常小心启动带有首先指令组的VM。初始阶段，操作系统（还有一些应用也是一样）会向CPU询问CPU支持的指令。然后必要时，他们会使用所有可用的指令。因此，要非常小心选择暴露给客户机的CPU型号。

QEMU可以支持暴露给客户机的集中虚机类型。这是由大量的兼容性代码组成的，这些代码支持多种暴露给客户机的设备。保持QEMU的bug修正和特征的持续改进是必要的，因为保持与以前QEMU版本的bug兼容性可以确保迁移持续进行。版本管理的另外一方面，就是多个ROM版本，通过QEMU设置来保证对客户机可用，例如BIOS, iPXE ROMs等。这些ROM一定要在源主机和目标主机上都兼容。

在线迁移的阶段

- Live migration happens in 3 stages 在线迁移分3个阶段
- Stage 1: Mark all RAM dirty 第一个阶段：将所有RAM都标记脏
- Stage 2: Keep sending dirty RAM pages since last iteration 第二个阶段：持续不断发送脏RAM
    - stop when some low watermark or condition reached
    - 当达到一些低水印或者条件时停止
- Stage 3: Stop guest, transfer remaining dirty RAM, device state
- Continue execution on destination qemu
- 第三个阶段：停止运行客户机，将剩余脏RAM，设备状态转移过去
- 在目标主机QEMU上执行

我们来讨论真实的迁移过程。这个部分指的是以上图表中中间和最左边部分。这个迁移过程分三个阶段进行。前两阶段处理图表中灰色区域，最后一个阶段灰色区域和最左边区域同时工作。

我们从第一阶段开始，将灰色区域每一页标记“脏页面”或者“需要被迁移”。然后我们将所有标记为脏页面的发送给目标主机，这就是第一步。

记住一件事，我们前面指出，KVM为用户空间提供什么服务？用户空间的能力，例如QEMU，也就是找出最后一次数据被请求过的更改过的页面能力。正是这样，我们才能持续不断发送自上次迭代后那些经过客户机更改过的页面。

这就是第二阶段。我们会一直停留在这个阶段直到达到某个条件。我们随后将会讨论这些条件。在此指出，这个阶段就是整个在线迁移过程最耗费时间的阶段。

第三个阶段就是过渡到非在线或者离线状态迁移。实际上客户机是暂停的，不再进行任何行动。然后我们将所有剩余脏RAM和设备状态转移过去，这些设备状态也就是图表中最左边区域。

灰色区域对于QEMU就是不透明数据，也要转移过去。这个设备状态是通过QEMU的特殊代码处理，稍后讨论这点。

在线迁移状态

- Live migration happens in 3 stages 在线迁移分3个阶段
- Stage 1: Mark all RAM dirty <ram_save_setup()>第一个阶段：将所有RAM都标记脏
- Stage 2: Keep sending dirty RAM pages since last iteration <ram_save_iterate()>第二个阶段：持续不断发送脏RAM
-
    - stop when some low watermark or condition reached当达到一些低水印或者条件时停止
- Stage 3: Stop guest, transfer remaining dirty RAM, device state <migration_thread()>第三个阶段：停止运行客户机，将剩余脏RAM，设备状态转移过去
- Continue execution on destination qemu 在目标主机QEMU上执行

控制每个迁移阶段的QEMU源代码功能名称会在以上图表中蓝色区域提及到，“迁移线程”功能是控制整个迁移过程，并且可以调用其他功能，所以我们从这开始理解整个迁移代码。

EndingStage 2 (or Transitioning from Live to Offline State)

- Earlier之前

    - 50 or fewer dirty pages left to migrate 50个或者更少的脏页面等待迁移
    - no progress for 2 iterations 没有2次迭代过程
    - 30 iterations elapsed 3 0个迭代消逝
- Now当前
    - involves knowing # of pages left and bandwidth available 包括了解脏页面剩余数量和可用带宽
    - admin-configurable downtime (for guests) 客户机admin-configurable故障时间
    - host policies: like host has to go down in 5 mins, migrate all VMs away within that time 主机策略：如果主机要在五分钟内停机，就要将所有VM在这个时间内迁移完成。

当从第二阶段到到第三阶段过渡时，要做一个很重要的决策：客户机在第三阶段会暂停运行，所以在第三阶段要尽可能的将少迁移页面，来减少停机时间。

QEMU的第一次迁移实施是很简单的：如果在第二阶段有50个或者更少的剩余脏页面需要迁移，将会过渡到第三阶段。或者当一个特定数量的迭代消逝，且不再有任何程序迁移少于50个脏页面，也会出现如上情况。

开始可能正常运行，但是随后就会出现一些新的限制。用户在KVM上运行负载时，做迁移必须要提供一些SLA，包括最大可接受停机时间等。所以我们要增加一些可调代码，以便达到客户机和主机都接受的从第二阶段过渡到第三阶段的条件。

客户机管理可以具体阐述最大可接受的停机时间。在第二阶段代码，我们会在每个迭代来检查有多少页面被客户机标记脏。会检查花费多长时间来转换一个页面，以便来设定一个预估的网络带宽。在这个预估带宽和当前迭代的脏页面数量，我们可以计算出花费多久来转化剩余页面。如果在可接受或者设置的停机时

间限制内，我们过渡到第三阶段是没有问题的。否则我们会继续停留在第二阶段。当然，还有其他的一些可调的因素会不断地让我们停留在第二阶段。

也有一些主机管理的设置可调因素。曾在2013年KVM论坛上，一个日本客户在一个座谈会上展示了一个很有意思的用例，这个用例描述了这个可调因素的必要性。日本数据中心接到地震警告，在这种情况下，数据中心会在一段时间内断电，大约从警告开始的30分钟内。服务提供者质优这30分钟将所有的VM从一个数据中心迁移到另外一个。否则所有的VM都将因为断电而宕机。在这种情况下，客户机SLA优先级下降。

QEMU的其他迁移代码

- General code that transmits data 迁移数据的总代码
    - tcp, unix, fd, exec, rdma
- Code that serializes data 使数据序列化的代码
    - section start / stop
- Device state 设备状态

QEMU中还有更多的与迁移有关的代码：有些代码是用来发送/接受迁移数据：TCP或者UNIX包，本地文件说明符，或者RDMA。也有exec功能，来自QEMU的数据，在被发送到目的地前，会被传输到其他进程。这对于传出数据压缩，加密都很有用。在目的端，也需要进行解压缩或者解加密进程。对于UNIX包，fd或者exec-based协议，需要更高层的程序来管理两端的迁移。Libvirt就是这个程序，我们可以依赖libvirt的能力来控制这个迁移。

当然也有与数据序列化相关的代码，这些代码可以指示页面何时何地开始，暂停等等。
也有与设备相关的代码，接下来我们要讨论到。

VMState

- Descriptive device state 描述性设备状态
- Each device does not need boilerplate code每一个设备不需要样本文件代码
- Each device does not need identical save and load code每个设备不需要完全相同的保留和加载代码

- Which is easy to get wrong这样容易出现错误。

设备状态，在以上图标中最左边区域，对每一个设备都会有详细的介绍。这个块设备将列出清单，计算仍需要被磁盘擦掉的动态数据。同时，这个状态的一部分就是客户机配置或者选择使用这个设备动作。网络设备也有自己的状态。几乎所有的设备都暴露给客户机，都有需要被迁移的状态。这个设备状态的迁移在第三个阶段发生，也就是在所有脏RAM传送结束之后。

每一个设备都必须控制好发送或者接收自己状态。这意味着有很多围绕QEMU所有设备的代码副本。VMstate就是一个新被添加的基础设施，将发送和接收部分的设备孤立出来。下面将用一个例子解释。

VMStateExampleVMState例子

- e1000 device e1000设备
- e482dc3ea e1000: port to vmstate
- 1 file changed,
- 81 insertions(+),
- 163 deletions(-)

将e1000设备从旧的处理状态做迁移，转换到VMState。需要插入81条命令，删除163条。这样就成功了。

VMStateExample (before)之前

```
-static void
-nic_save(QEMUFile *f, void *opaque)
{
   E1000State *s = opaque;
   inti;

   pci_device_save(&s->dev, f);
   qemu_put_be32(f, 0);
   qemu_put_be32s(f, &s->rxbuf_size);
   qemu_put_be32s(f, &s->rxbuf_min_shift);
```

这是一个命令执行部分，涉及到以上所说的旧代码移除。这个nic_save()功能是“保留”——传送中——在目的端需要所有的状态都能正确操作，独立地传送每一个条目状态。在此我们看到了32位整数结构的被传送。

与nic_load()相对应的功能会按照相同的次序加载这个状态。
很容易看到如何出错，哪些副本对于设备是不必须的。同时，开发者们一点也不喜欢这样写代码。

VMStateExample (after)之后

```
+static constVMStateDescription vmstate_e1000 = {
+ .name = "e1000",
+ .version_id = 2,
+ .minimum_version_id = 1,
+ .minimum_version_id_old = 1,
+ .fields = (VMStateField []) {
+ VMSTATE_PCI_DEVICE(dev, E1000State),
+ VMSTATE_UNUSED_TEST(is_version_1, 4), /* was instance id */
+ VMSTATE_UNUSED(4), /* Was mmio_base. */
+ VMSTATE_UINT32(rxbuf_size, E1000State),
+ VMSTATE_UINT32(rxbuf_min_shift, E1000State),
```

我们看到命令执行后的一部分代码，添加了新的VMState方式，保留和修复设备状态。这只是一个描述性的结构，来描述设备状态的组成。所有样本文件代码在其他地方处理过，都显而易见地添加到每一个设备上。

UpdatingDevices更新设备

- Sometimes devices get migration-breaking changes有时设备会破坏性迁移改变
- One idea is to bump up version number增加新版本号
  - Adds dependencies from higher versions to lower ones在低版本上依次添加高版本号
  - Difficult to cherry-pick fixes to stable / downstreams很难择优选择修复
- Another is to introduce new subsection另一个是介绍新的小节

-

作为规律的开发过程，会发现bug并修复它，新特征也是这样添加的。当设备状态有了新变化，很多因素都要考虑到，来保持迁移后的兼容性。也就是，从老QEMU版本迁移过来的在新QEMU版本上运行的兼容性。新QEMU版本应该继续接受即将从老的QEMU版本迁移过来的数据流。尽管状态改变了。

控制好这样的变化的一个典型的办法就是靠给每一个设备进行版本控制，传送过程中就改好版本。源QEMU不需要担心目的端运行什么版本。这只取决于目的端QEMU来翻译传送过来的各种格式。

与目的端QEMU同样的加载功能是，检查传送过来的是什么版本的设备状态。依赖版本管理，传过来的数据被存储到目的端设备结构的相应状态变量中。这意味着，所有版本相关的复杂事务，都会存在迁移过程的目的端。

当然，目的端比源QEMU更低版本的QEMU不会知道传送过来的数据，迁移过程在这种情况下会失败。

但是这种版本管理机制有一个问题：试想版本2的e1000设备状态。现在我们添加一个新特征，需要给状态再添加一些东西，变为版本3.如果我们在版本2的状态中检测到一个bug，我们会修复它，更新到版本4。然而，修复后变为稳定版本发行出来，或者不想发行新特征，但是想要补丁修复就难了。没有办法补丁修复，并且保持与上一个版本号的连续性。变为版本4不正确，因为版本3还没打补丁。

需要一个新的机制来解决这个困境，下一节将会讲到，让我们来看一个例子。

SubsectionExample

- commit c2c0014 pic_common: migrate missing fields

  执行c2c0014 pic_common：迁移丢失的区域。

```
VMSTATE_INT64(timer_expiry,
APICCommonState), /* open-coded timer state */
VMSTATE_END_OF_LIST()
+ },
+ .subsections = (VMStateSubsection[]) {
+ {
```

```
+    .vmsd = &vmstate_apic_common_sipi,
+    .needed = apic_common_sipi_needed,
+ },
+ VMSTATE_END_OF_LIST()
}
};
```

我们看到一个新部分添加到apic代码的vmstate数据中。这个所需功能会在源端评估。如果结果是真的（也就是客户机处于那一小部分需要被传到目的端的状态）在vmsd部分中vmstate信息将被传送到目的端。
这个消除版本管理的需要，最优选择更简单些，并且能在多个QEMU版本中兼容。

最近变化

- Guests have grown bigger客户机变大
    - Means active guests keep dirtying pages 意味着运行中的客户机要不断保持标记脏页面
    - Means a lot of time spent transferring pages 意味着要花费更多时间传输页面
    - More RAM 更大RAM
    - More vCPUs 更多vCPU

现在我们讨论一下QEMU在迁移代码中如何运行。KVM已经成熟了，更多地客户用它在他们的KVM虚拟化层来处理企业加载。我们看到更大的客户机：分配更多RAM，更多的vCPU暴露给客户机。大RAM客户机意味着有更多的数据会在第二阶段被传送，反过来，意味着随之而来的停机时间要求变得更加困难实现。更多vCPU意味着客户机要做更多的工作，更多的RAM区域要被标记脏，导致第二阶段时间变长。

QEMU最近的一些新特征消失了，有助于以下这些情况：

NewFeatures新特征

- Autoconverge 自动汇聚
- xbzrle

- migration thread 迁移线程
- migration bitmap
- rdma
- block migration 块存储迁移

自动汇聚：这个设置可以暂停一些vCPU，通过降低迁移速度，客户机在第二迁移阶段不会进行太多过程。这样就给一些脏页面机会，来减少我们过渡到第三阶段的低水印条件。暂停客户机vCPU可以有几种方式：重新启动主机客户机线程（每一个vCPU在主机就是一个线程）或者限制主机CPU时间，或者在QEMU代码中不将客户机vCPU列入计划。

Xbzrle：这个特征是将上一个迭代的所有发到目的端的页面进行缓存。对于每一个被发送的页面，我们将当前页面与缓存页面做比较，只有在页面比较出来不同的字节才被发送过去。这样能够很大程度地减少占用网络负载的数据量，可以加速迁移过程。

迁移线程：起初，QEMU代码为QEMU所有活动只占用一个线程，为客户机vCPU分配其他线程。这意味着任何QEMU任何一个活动都会阻碍其他活动进行。在迁移用例中，当发送和接收数据时，QEMU的其他活动都无法进行，这也意味着与QEMU交互的应用不能查询QEMU迁移进行到什么程度了。缺少报告对于管理来说体验很不好，也就是说要在第二阶段等上一个迭代结束后，才能知道这个过程是不是完成了。将迁移代码分配到与之配备的线程上，QEMU在这个迁移线程上运行，这样可以减少瓶颈了。

迁移：一个字节过去用来追踪一个脏页面。对于大RAM客户机来说，意味着位图会变得很大，甚至比CPU缓存更大，第二阶段的每个迭代的页面操作都会变得很慢。另一个缺点就是当主机在低内存时进行迁移，由于要给迁移元数据分配更多的RAM，导致主机运行更慢。我们现在使用一个比特代表每一个客户机页面，减少脏页面位图，这样CPU缓存也不会占用很多，整个过程也不会很慢。

RDMA：使用像infiniband设备一样，在源端和目的端主机的快速互联，可以确保迁移数据比以太网传送的更快。RDMA代码还很新，没有经历过充分的测试，所以在这个领域，也有潜在贡献者在寻找其他技术提升其性能。

块存储迁移：除了迁移这些状态，一些客户需要使用非共享块存储。这样，存储也要随着主机状态一起被迁移。块数据很大，要在网络带宽和需要迁移的时间上花费很大代价。

Stuffthat’s lined up

- postcopy预先拷贝
- debuggability排障能力

迁移代码中有很多正在进行的任务，其中一个就是预先拷贝，这样会使迁移停机时间变得非常短。

我们之前描述的迁移过程就是预先拷贝模型。在切换到目的端客户机运行之前，所有数据先拷贝到目的端主机。在预先拷贝模型中，要尽块切换到目的端，这意味着我们要将设备状态，一个迭代脏RAM传送过去，然后切换到运行目的端。当客户机引用这个RAM的某些页面，而这个页面在目的端不存在，远程页面容错会从源主机中把页面抓取过来传送到目的端。这样会很容易理解为什么这个方法在聚合速度上很快。当然，也会引起其他的一些错误：如果一个主机坏掉，或者两个主机之间的网络连接断掉，整个客户机就会丢失，两个主机就不会再通信了。对于预先拷贝文件机制，源主机的丢失，意味着客户机永远丢失。

迁移过程的排障能力不是很好：有线格式描述很差，迁移失败会给出这样的信息“设备加载失败，迁移失败”不会给出有关设备加载失败的位置和原因。潜在贡献者可以找到一些技术将这个优化的更好些，这也是一个学习代码的好机会。

Futurework未来任务

- Finish vmstate conversion完成vmstate转换
- self-describing wire format有线格式自描述

像完成vmstate转换这样的任务还没有人开始着手。不幸的是，不是所有的设备都转化成vmstate。最大的一个就是virtio设备。这样做很有必要，以便于所有的东西都有统一的代码。也有一个检查程序，来检查不同QEMU版本之间的迁移兼容性。这种程序只以vmstate数据格式工作，而virtio设备目前还不能实现。

同时，如上一部分提到的，自描述功能的有线格式有助于排障，也有助于监控各种抵消掉的数据流等等。这个目前还不可能实现。

这只是QEMU-KVM虚机在线迁移的概览。QEMU归档清单里有几个细节，包括各种优化性能指标，以及一些各自指标的特征讨论。在以下章节我们会继续讨论到。

转自：QEMU-KVM虚机动态迁移原理