基于内存混合复制方式的虚拟机在线迁移机制

陈 阳 怀进鹏 胡春明
(北京航空航天大学计算机学院 北京 100191)

摘 要 虚拟机 VM(Virtual Machine)在线迁移技术(Live Migration)充分利用虚拟化技术的灵活性和封装性,有利于实现大规模虚拟化环境中负载的动态调整以及应用的灵活部署.已有的在线迁移机制存在内存迭代收敛、迁移数据冗余以及客户操作系统不透明等问题.文中结合内存推送复制以及按需复制两种方式,提出了基于内存混合复制方式的 VM 在线迁移机制 HybMEC,以实现对 VM 运行状态的快速迁移.基于 KVM 虚拟机监控器,实现了 HybMEC 原型系统.多种不同类型应用负载下的实验结果表明,HybMEC 能够高效地支持虚拟化环境中低开销、低延时的 VM 在线迁移,并在总迁移时间、内存同步数据量以及 VM 停机时间等方面具有显著的性能提升.

关键词 虚拟机;虚拟机监控器;在线迁移;按需复制;预取页
中图法分类号 TP302 **DOI 号**: 10.3724/SP.J.1016.2011.02278

Live Migration of Virtual Machines Based on Hybrid Memory Copy Approach

CHEN Yang HUAI Jin-Peng HU Chun-Ming
(*School of Computer Science and Engineering, BeiHang University, Beijing* 100191)

Abstract The live migration technology of virtual machine, fully leveraging virtualization's flexibility and encapsulation abilities, is very helpful for dynamic workload balancing and flexible application deployment in the large-scale virtual machine environments. However, the existing live migration mechanisms have to be in the face of some problem, such as convergence of iteration, redundance of copying, and guest OS transparence. This paper discusses the application of combining memory pushing copy with on-demand copy approaches, and proposes a live migration mechanism based on hybrid memory copy approach, named HybMEC, in order to accelerate VM state migration. Base on KVM virtual machine monitor, we have implemented the HybMEC prototype system. The evaluation results under various VM workloads show that HybMEC is able to efficiently support VM live migration with lower resource consumption and latency, and to achieve significant improvements over existing mechanisms by the total migration time and the volume of data synchronized over the network.

Keywords virtual machine; virtual machine monitor; live migration; on-demand copy; pre-paging

1 引 言

当前云计算作为一种新型网络应用模式,已逐渐成为科学界和工业界研究的热点主题.云计算实现了资源、平台以及软件的服务化,并通过计算机网络向用户提供按需、灵活、可伸缩的计算和存储资源.其中,基础设施即服务 IaaS(Infrastructure as a

收稿日期:2011-06-14;最终修改稿收到日期:2011-10-27.本课题得到国家"八六三"高技术研究发展计划项目基金(2011AA010500)、国家"九七三"重点基础研究发展规划项目基金(2011CB302602)和国家杰出青年基金项目(90818028,91018004)资助.**陈 阳**,男,1983 年生,博士研究生,主要研究方向为虚拟化技术、分布式计算. E-mail: chenyang@act.buaa.edu.cn. **怀进鹏**,男,1962 年生,教授,博士生导师,中国工程院院士,主要研究领域为网格计算、分布式计算、网络安全. **胡春明**,男,1977 年生,博士,副教授,主要研究方向为虚拟化技术、分布式计算、网格计算.

Service)，例如 Amazon EC2①、Eucalyptus[1]，更被认为是云计算的先行者[2]. IaaS 以“按需付费”的模式向用户提供虚拟化基础设施，有效地提高了物理资源利用率以及应用程序的可用性，同时简化了用户对于 IT 基础设施配置和运维管理，大幅度地降低了硬件投资成本. 在 IaaS 高灵活性和高可扩展性的背后，虚拟机 VM 技术(Virtual Machine Technology)[3-4]是其实现的核心技术. 后者通过软件抽象将单个物理机资源划分成多个相互隔离的虚拟资源，实现了多个操作系统实例对单个物理资源的复用. 然而，VM 在线迁移技术更是 VM 技术灵活性和可用性的重要应用和体现. 在线迁移技术实现了 VM 运行状态(即操作系统运行状态)在分布、异构的物理运行环境中快速的平移. 整个迁移过程中，VM 的运行状态只在极小的时间间隔内被中断，这使得 VM 应用对外表现出“不间断”的可访问性. 目前，VM 在线迁移技术广泛地应用于云计算虚拟化数据中心(virtual data center)环境中的运维管理，例如，动态负载均衡和资源管理、物理设备能耗调整以及在线的设备维护(online maintenance)等.

实现 VM 内存状态在源、目的宿主机(host machine)之间的高效复制是 VM 在线迁移技术的关键所在. 目前，主流虚拟化平台均提供 VM 在线迁移功能，并且在实现上普遍采用了基于内存预拷贝 Pre-copy 机制[5-6]. 但是，Pre-copy 机制受其自身特性的约束，在一些情况下并不能取得理想的迁移效率. 例如，在低带宽网络环境中或对于大内存 VM，Pre-copy 机制很可能会由于迭代收敛性问题，导致迁移过程中 VM 应用性能出现明显的下降. 而对于访存密集型应用，由于 VM 内存更新频繁，Pre-copy 机制需要经历低效的内存迭代同步才能完成内存一致性状态的复制. 过长的迭代过程导致迁移总时间被延长，造成迁移过程对物理资源的长时间占用. 此外，程序访存的局部性还会造成 Pre-copy 机制在内存迭代复制过程中一些内存数据被多次重复地复制. 这种迁移数据的高度冗余性从另一方面导致了迁移过程对物理资源不合理的消耗.

云计算系统服务于大量企业或个人用户，而系统中有限的物理资源需要在众多用户之间实现共享，因此单用户往往不允许以独占的方式使用系统分配的物理资源. 但是，现有的 VM 在线迁移机制在资源受限的环境中，常表现出低效的性能甚至出现迁移失败的情况. 另外，在物理资源共享的环境中，当前用户所执行迁移的操作，可能会损害其他用户对物理资源的需求；或者迁移操作受限于被其他用户占用的物理资源. 因此，在线迁移机制在实现 VM 执行环境快速切换的同时，更需要考虑迁移过程本身对于物理资源的占用和消耗.

针对现有在线迁移机制性能和实现上的不足，以及云计算环境中用户物理资源共享且受限的特点，本文提出基于内存混合复制方式的 VM 在线迁移机制 HybMEC(Hybrid MEmory Copy). 即 HybMEC 在迁移过程中充分结合内存推送复制和按需复制两种方式实现 VM 内存状态快速、高效的同步. 其中，推送复制是指源宿主机主动向目的宿主机复制内存数据；而按需复制是指目的宿主机适时地向源宿主机请求某一部分内存数据. HybMEC 以缩短总迁移时间，避免迁移过程对物理宿主机的长时间依赖为首要目标. 同时，HybMEC 还尽可能地减少迁移过程对物理资源的消耗，同时降低 VM 应用在迁移过程中的性能损失. 总体上看，HybMEC 将内存同步的过程严格划分为全内存同步、内存位图同步以及脏内存同步 3 个阶段. 其中，全内存同步阶段，HybMEC 在保持 VM 运行的前提下，利用源宿主机主动推送的方式将 VM 完整的内存数据复制到迁移目的端. 该阶段保证了在 VM 切换到目的宿主机上运行时，绝大多数内存数据经过一次复制已经驻留在在目的宿主机本地. 在全内存同步结束后，HybMEC 不陷入脏内存的迭代复制，而是迅速地复制脏内存位图数据并在目的宿主机上恢复 VM 执行，随后真正进入脏内存同步阶段. 一方面利用虚拟机监控器 VMM(Virtual Machine Monitor)在目的宿主机端捕获 VM 对脏页面的访问，HybMEC 通过网络缺页处理从源宿主机端按需地复制某一部分未同步的脏页面数据；另一方面为了加速脏页面的同步，HybMEC 定时地从源宿主机上推送剩余的脏页面数据. 因此，从迁移过程上看，HybMEC 分阶段地利用推送复制和按需复制实现了内存状态的高效同步；而从脏内存数据的角度，HybMEC 充分结合按需复制及定时推送的优势，实现了对脏内存数据的快速复制.

由于脏页面的按需复制要求中断当前 VM 的正常运行后才能开始执行网络缺页处理. 因此，频繁的按需复制势必导致迁移过程中 VM 运行性能出现剧烈抖动，并延长 VM 应用的执行延时. 为了避

① Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2

免频繁地陷入由按需复制引起的缺页中断处理，根据访存的局部性和顺序性原理，HybMEC 在该类型缺页中断处理中采用了预取页(Pre-paging)机制. 即从源宿主机端按需地复制一个脏页面时，HybMEC 会“捎带”地预先复制该缺页页面的邻居脏页面. 这样，在很大程度上 HybMEC 保证了 VM 后续访问邻居脏页面时，这些页面已经驻留在目的宿主机本地，而避免再一次引发高延时的网络缺页处理. 另外，为进一步提高内存同步的效率，一方面，HybMEC 主动避免对空闲页面的按页面大小(4 KB)进行复制；另一方面，HybMEC 禁止空闲脏页面引起按需复制处理，转而通过 VMM 为空闲的脏页面直接分配本地内存页.

我们在 KVM 虚拟化平台上实现了 HybMEC 原型系统，并通过多种不同类型的应用负载验证了 HybMEC 机制的高效性和可用性. 实验结果表明，相比于 Pre-copy 机制，HybMEC 不仅减少了迁移过程的同步数据量、降低了迁移过程造成的 VM 运行性能损失，而且缩短了 VM 停机时间以及总迁移时间. 特别是对具有访存密集型应用的 VM，HybMEC 在上述的各方面性能上均有显著的提升.

本文第 2 节将详细介绍 VM 在线迁移机制基本背景以及 HybMEC 机制的原理和方法；第 3 节介绍 HybMEC 在 KVM 上的具体实现细节；第 4 节介绍原型系统的实验结果及分析；第 5 节介绍相关工作对比；最后第 6 节小结全文.

2 HybMEC 实现原理

2.1 VM 在线迁移原理

本质上，在线迁移机制实现了 VM 运行状态通过计算机网络在物理宿主机之间的高效复制. 其中，运行状态包括 VM 的虚拟处理器 VCPU 寄存器、内存以及外部资源设备状态(即外部磁盘文件系统和网络连接状态). 在局域网 LAN 环境中，VM 外部资源设备状态一般可通过配置网络共享存储设备(NAS、NFS 等)以及主动汇报网络拓扑变化，解决迁移过程中 VM 外部设备的状态一致性问题[5-6]. 因此，解决内存状态的高效同步成为 VM 在线迁移的关键问题. 通过计算机网络，将 VM 内存状态从一台物理宿主机复制到另一台物理宿主机可通过以下 3 种方式[6-7]：

(1) 停机拷贝. 源宿主机暂停 VM，将 VM 内存状态完整地复制到目的宿主机；目的宿主机正确加载内存状态后启动 VM.

(2) 目的端按需复制. 源宿主机暂停 VM 运行，VM 内存状态驻留在源宿主机上；随后 VM 在目的宿主机上恢复运行，并按需地从源宿主机获取内存页面.

(3) 源端推送复制. 源宿主机保持 VM 持续运行，并主动将 VM 内存状态复制到目的宿主机.

现有的迁移机制中，基于内存预拷贝(Pre-copy)的迁移机制结合了源端推送复制和停机拷贝两种方式；而基于内存后拷贝(Post-copy)的迁移机制[8]结合停机拷贝和目的端按需同步两种方式. 以 Pre-copy 机制为例，其完整的迁移流程如图 1(a)所示. 迁移初始，VMM 将所有的 VM 内存页面标记为“脏页面”；接着，迁移过程开始进入多轮的内存迭代同步过程. 值得注意的是，由于所有页面都被置为“脏页面”，第一轮内存同步需要复制全部的 VM 内存数据；而后续的每一轮同步只需要复制上一轮同步过程中被 VM 修改的脏页面数据. 经过若干次迭代后，如果剩余脏页面数小于预设最小值或迭代次数大于预设最大值，Pre-copy 机制终止迭代，并主动暂停源宿主机上的 VM，将剩余的脏页面以及其它虚拟设备状态推送至目的宿主机. 最后，目的宿主机接收并加载最后的状态数据后立即恢复 VM 执行，同时向源宿主机确认迁移完成.

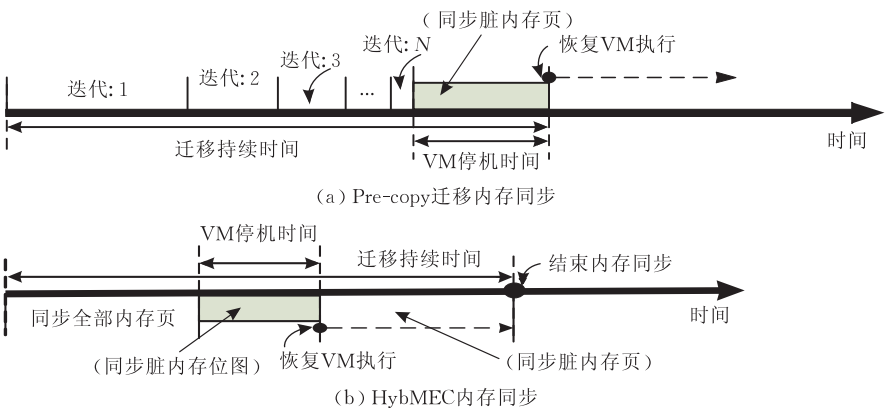

图 1 Pre-copy 与 HybMEC 内存同步过程对比

在迁移过程中，为了获得 VM 在某一时刻的完整运行状态（VCPU 寄存器、内存以及其它虚拟设备状态），VM 正常运行需要在一段时间内被中断. 如果该时间间隔足够小，迁移机制就能保证 VM 与外部的网络连接在此期间不发生超时，并避免 VM 应用对外部表现出明显的不可访问. 例如，Pre-copy 机制在结束内存迭代同步后，立即暂停源宿主机上 VM 的运行，并在随后的某一时刻恢复 VM 在目的宿主机上的运行. 理想情况下，Pre-copy 机制能够保证该时间间隔介于几十毫秒到 1 秒之间. 我们将该时间间隔定义为 VM 停机时间，它等于源宿主机暂停 VM 时刻与目的宿主机恢复 VM 执行时刻的间隔，即 VM 分别在源和目的宿主机上切换运行状态的时间间隔. 对于 Pre-copy 机制，VM 停机时间主要由停机时刻剩余状态数据的大小及可用物理网络带宽决定. 因此，如果剩余脏页面数据过大或者物理网络带宽过小，那么 VM 停机时间将受此影响而被延长.

在实现上，VM 在线迁移机制首先需要保证透明性，即迁移过程对 VM 以及其内部应用透明. 同时，迁移机制还需要权衡各方面的性能以及迁移过程本身对物理资源的占用.

（1）VM 停机时间. VM 被暂停执行的时间间隔；直接反映了 VM 不可访问的时间.

（2）总迁移时间. 从迁移开始到结束的持续时间；决定了迁移过程对物理资源的依赖时间，其中物理资源主要包括物理网络带宽、宿主机 CPU 资源.

（3）内存同步数据量. 整个迁移过程中复制的内存页面数量；直接反映了迁移过程对物理网络带宽的占用情况.

（4）VM 应用性能损失. VM 内部应用执行延时或对外表现出的性能抖动；直接反映了迁移过程对 VM 应用性能的影响.

虽然 Xen[3]、VMware[5] 及 KVM[9] 等虚拟化平台均提供了基于上述 Pre-copy 机制的在线迁移功能，但是由于依赖基于源端推送复制的内存迭代同步，Pre-copy 机制并不能保证稳定的迁移性能. 例如，潜在的迭代收敛性问题，即当网络传输速率小于 VM 更新其内存的速率时，迁移过程经过多轮迭代复制后剩余的脏页面仍无法收敛于预设的最少脏页面数的问题. 在该情况下 Pre-copy 机制需要被迫终止内存迭代同步，立即暂停 VM 运行并进入停机拷贝阶段. 但此时，大量未同步脏页面数据驻留源宿主机上等待停机时间内的复制，造成 VM 停机时间被延长. 其次，内存迭代同步方式直接造成迁移性能对剩余脏内存收敛速率的过分依赖. 而后者在不同的 VM 应用负载下往往表现出强烈的不确定性. 最后，访存的局部性使得 Pre-copy 机制在上一次迭代中复制的内存页面很可能在后续的迭代中被再次修改，并且修改只局限于该页面上的某一小部分数据. 因此，内存迭代同步造成了迁移数据具有很高的冗余性. 上述数据的冗余性必定会导致迁移过程对物理网络带宽不必要的消耗和占用.

2.2　HybMEC 原理

针对现有迁移机制的不足，如图 1（b）所示，HybMEC 在迁移过程中分阶段地利用源宿主机推送复制以及目的宿主机按需复制两种方式，实现对 VM 内存状态的高效同步. HybMEC 将整个内存同步过程划分为 3 个阶段：

（1）全内存同步. HybMEC 按页为单位将整个 VM 内存数据从源宿主机复制到目的宿主机，并在结束时刻暂停 VM 运行. 由于该过程中 VM 在源宿主机上仍保持运行，因此 HybMEC 需要主动地标记所有被更新的脏页面.

（2）内存位图同步. 源宿主机推送上一阶段里记录“脏页面”的位图数据 dirty_bitmap；目的端宿主机根据接收到的 dirty_bitmap 数据设置 VM 相应页面的状态.

（3）脏内存同步. 目的宿主机恢复 VM 执行，HybMEC 利用 VM 访问未同步的脏页面产生的缺页异常，从源宿主机端按需地复制脏页面数据；同时，源宿主机定时向目的宿主机端推送未同步的脏页面. 当所有脏页面复制完成后，在线迁移过程结束.

相对于 Pre-copy 机制，HybMEC 从根本上避免了内存迭代同步. 同时，在 VM 停机时间内，HybMEC 只复制内存位图数据 dirty_bitmap，减少了 VM 停机时间内复制的数据量，缩短了 VM 运行被中断的持续时间. 其次，由于在整个迁移过程中只有脏页面数据需要额外的一次复制，而绝大多数的内存数据只需要经过一次的复制即完成同步. 因此，HybMEC 大幅地减少迁移过程中同步数据量，同时降低对于宿主机物理资源以及物理网络带宽的占用. 最后，虽然每一次的脏页面按需复制需要强制中断 VM 的运行，但是相对于 Post-copy 机制全内存按需复制的方式，HybMEC 只按需复制部分脏页面数据，这在很大程度上降低了缺页处理造成的性能损失，避免了 VM 应用性能出现剧烈抖动，同时也

缩短了迁移持续时间.

完整的 HybMEC 迁移过程如图 2 所示. 其中,步骤 VI-VIII 将循环执行直至所有 VM 内存同步完毕. 步骤 I 中,源宿主机发起迁移操作并向目的宿主机复制完整的 VM 内存数据,同时在 dirty_bitmap 中标记复制过程中被修改的页面;随后的步骤 II 中,源宿主机暂停 VM 执行. 此时 VM 的内存、VCPU 寄存器以及其它虚拟设备状态在源宿主机上被冻结. 在步骤 III 里,源宿主机向目的宿主机复制 dirty_bitmap、VCPU 寄存器以及其它虚拟设备状态. 另一端,目的宿主机执行步骤 IV,根据 dirty_bitmap 设置脏页面的"缺页"状态,同时加载 VCPU 寄存器以及其它虚拟设备状态;最后恢复 VM 执行. 随后,VM 在目的宿主机上正常运行的过程中访问"缺页"状态的脏页面,产生缺页异常被 VMM 捕获(步骤 VI)并通过目的宿主机向源宿主机发送缺页请求(步骤 VII). 步骤 VIII 中,源宿主机根据请求消息中缺页信息以及本地 dirty_bitmap 定位脏页面,并以该页面数据响应缺页请求;而目的宿主机接收该缺页数据后,将其写入缺页地址指示的内存页,并立即恢复 VM 执行.

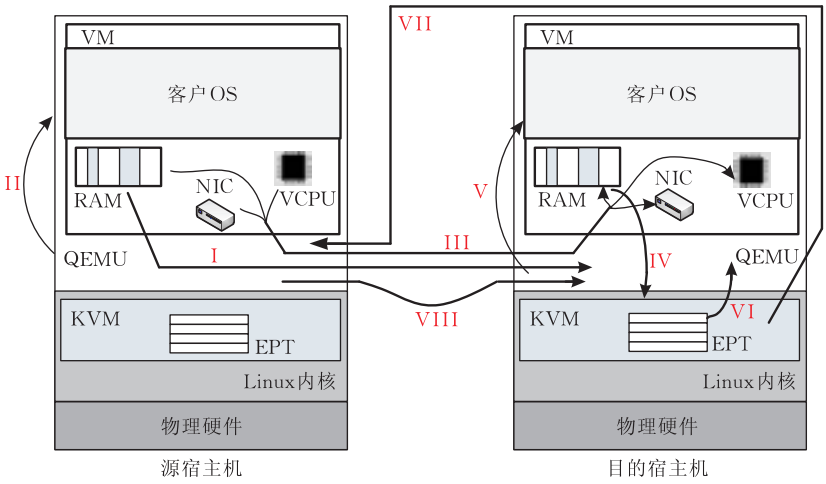

图 2 HybMEC 迁移流程

在上述的迁移过程中,步骤 I-II 对应于全内存同步阶段,完成了大部分内存数据的一次性复制;而步骤 III-IV 对应内存位图同步阶段,除了 dirty_bitmap 数据,源宿主机同时向目的宿主机复制 VCPU 寄存器以及除内存外的其它虚拟设备状态数据. 最后,步骤 VI-VIII 对应脏内存同步阶段. 当所有脏页面完成同步后,目的宿主机上 VM 的运行不再依赖源宿主机,HybMEC 断开两端宿主机之间用于内存同步的网络连接,并释放源宿主机上为迁移 VM 分配的物理内存.

2.3 脏页面同步

VM 在目的宿主机端恢复运行,HybMEC 正式进入脏内存同步阶段后,由于程序访存的不确定性[10],VM 可能在一段时间内不再访问某一些脏页面. 这将造成依赖于"脏页面访问"驱动的按需复制不能快速完成所有脏页面的同步,同时也导致总迁移时间的不确定性. 因此,在该阶段里 HybMEC 利用脏页面定时推送来辅助脏页面按需复制,以避免脏页面的同步过程受限于 VM 对内存访问模式. 在进入脏页面同步阶段后,源宿主机上 HybMEC 按预设的时间间隔,定时地向目的宿主机推送未同步的脏页面数据. 这样,即使在最坏情况里,VM 不再访问任一脏页面或者访问任一脏页面之前该页面已经被预先复制,HybMEC 仍可以通过有限次的定时推送完成所有脏页面的同步.

每一次的脏页面按需复制都需要借助于网络缺页异常处理从源宿主机端复制指定的脏页面. 在该过程中,VM 需要被无条件地挂起,直至脏页面数据被加载后才能恢复运行. 因此,简单的"一次一页面"的响应模式将会恶化脏内存同步的效率,同时造成 VM 应用性能的损失. 因此,HybMEC 在源宿主机上维护一个同步窗口,适时地用脏内存数据填充该窗口,并以窗口大小为基本单位执行脏页面复制. 然而,频繁的脏页面按需复制势必会频繁地打断 VM 正常运行,导致 VM 应用性能出现抖动. 同时,由于按需复制的处理依赖于高延时的网络处理以及大开销的 VM 状态切换,频繁的脏内存按需复制势必还会影响 VM 应用的执行延时. 所以,为了避免频繁的缺页中断处理,缩短 VM 应用的执行延时,并加速脏内存的同步过程,HybMEC 在按需复制的缺页

处理中采用预取页机制.预取页机制最早由 Denning 提出,它利用程序的历史运行信息预测程序最近的工作集(working set)[11].该机制常被应用于程序内存页面的预先加载,例如磁盘数据文件的预先加载.根据应用访存的局部性和顺序性原理[10],当前被访问且发生缺页异常的脏页面的邻居页面很可能在后续的运行过程中被访问.如果该邻居也是脏页面且未被同步,那么 VM 在访问该邻居脏页面时必然会再一次引起网络缺页处理.但是,如果 HybMEC 在当前缺页响应时"捎带"上该邻居脏页面,那么就能直接地避免上述潜在的网络缺页处理.因此,当源宿主机在收到一次缺页请求后,HybMEC 利用缺页信息以及 dirty_bitmap 数据定位当前缺页页面,有选择地用"邻居"脏页面填充上述的同步窗口,并迅速响应该缺页请求.随后,当 VM 访问被"捎带"的脏页面时不再发生缺页异常,而转向对本地内存的直接访问.

2.4　内存缺页设置

在脏内存同步阶段,HybMEC 需要及时地捕获 VM 对任一脏页面的访问.如若不然,HybMEC 将错过了复制该页面数据的时机,很可能造成 VM 由于读写了错误的内存数据而出错甚至导致 VM 宕机.考虑到透明性需求,HybMEC 不能依靠 VM 访问脏页面时的主动通知来执行按需页面复制.因此,为了感知 VM 对任一脏页面的访问,HybMEC 需要根据内存虚拟化原理,在目的宿主机恢复 VM 运行之前预先为脏页面设置相应的缺页状态.同时,由于设置缺页状态的操作在 VM 停机期间执行,因此该操作的执行延时直接影响了迁移过程中 VM 停机时间.所以,HybMEC 需通过透明、高效的方式控制脏页面状态.

系统虚拟化中的地址转换相比于传统操作系统,额外引入了一层地址映射转换的工作.其中,客户机操作系统(简称,客户 OS)负责 VM 内部的线性地址到客户 OS 物理地址的映射;而 VMM 中内存虚拟化模块负责客户 OS 物理地址到真实机器地址的映射.VM 运行时,VMM 负责将由其维护的页表数据结构写入物理 MMU(Memory Management Unit)用于 VM 访存时的地址转换.因此,HybMEC 可通过控制该页表数据结构中相应表项的属性以实现对脏页面缺页状态的设置.

在迁移步骤 IV 中,目的宿主机在接收到 dirty_bitmap 数据后,HybMEC 根据 dirty_bitmap 中记录的脏页面客户 OS 物理地址对相应的页面设置缺页状态.虽然,利用内存虚拟化的纯软件实现影子页表技术 SPT(Shadow Page Table)[12],HybMEC 亦能实现对于任一 VM 内存页状态的控制,但是考虑到执行的效率及实现的复杂性,HybMEC 选择利用 EPT(Extending Page Table)硬件辅助功能[13-14]实现对脏内存页透明、高效的控制.这样,为一个脏页面设置缺页状态的操作,HybMEC 只需要根据该页面的客户 OS 物理地址查找 EPT,并将相应的表项上设置为"不存在"(non-present).随后在脏内存同步阶段,VM 访问"不存在"脏页面时,将产生缺页异常并自然地陷入 VMM,而被 HybMEC 捕获.

2.5　内存页复制优化

通常情况,VM 运行时并不会使用物理宿主机分配的全部内存,因此 VM 内存页中往往含有大量未使用的空闲页面(free page),并且这些页面一般被初始化为全零页(zero page).如果迁移过程中 HybMEC 完整地复制这些空闲页面势必会不必要地占用物理 CPU 及网络带宽资源.最近,Hines 等人[15]提出利用动态内存气泡机制(dyanmic self-ballooning)避免在 Post-copy 迁移过程中复制空闲的内存页面.其思想是,迁移过程中通过定时的气泡回收机制,将 VM 空闲页面换出主内存,从而一定程度上避免由于 VM 访问空闲页面引起网络缺页处理.但是,在 VM 在线迁移机制中引入动态内存气泡回收机制,一方面使得 VM 在线迁移与机制复杂的 VM 内存虚拟化紧耦合;另一方面要求客户 OS 主动支持内存气泡回收机制.

为了避免迁移过程中按照 4KB 大小复制空闲内存页,HybMEC 采用了简单而高效的方法.即在复制一个页面之前,HybMEC 首先判断该页面是否为空闲页.如果是空闲页面,HybMEC 只需要在迁移数据流中设置一个标志位指示该页面为空闲页;反之,HybMEC 向数据流中写入完整的页面数据.相比于复杂的内存压缩算法[16],上述的方法只针对空闲页面处理,在内存同步数据量和物理 CPU 资源消耗上做出较好的折衷.实验表明,上述主动避免空闲页面复制的方法能够有效地节省迁移过程占用网络带宽资源.

另外,为一个空闲脏页面执行按需复制的做法显然是低效而不可取的.这是因为空闲脏页面的按需复制不但造成 VM 发生不必要的状态切换,而且还使得迁移过程占用不必要的物理网络带宽.因此,迁移步骤 I 中,HybMEC 在 dirty_bitmap 数据结构中还标注了所有的空闲脏页面;迁移步骤 VI 中,当 VM 访问一个空闲脏页面而陷入缺页处理时,目的

宿主机上 HybMEC 不向源宿主机发送缺页处理请求，转而通过 VMM 直接从本地分配一个页面，并立即恢复 VM 运行.

3 HybMEC 实现

我们在 KVM 虚拟化平台(版本 KVM-86/88)上实现了 HybMEC 原型系统. KVM 以驱动模块的形式加载进入 Linux 内核，将 Linux 操作系统转换成为一个高效的虚拟机监控器. 同时，KVM 通过系统调用 ioctl 及内存共享实现与用户态进程 QEMU 交互. 后者负责 VM 初始化配置及关键虚拟设备模拟，例如 PCI 总线、网卡 NIC、IDE 硬盘等. 原型系统的实现如图 3 所示. 在用户态进程 QEMU 中，我们用 HybMEC 迁移控制器替换了 KVM 原有的基于 Pre-copy 迁移机制的迁移控制模块，同时增加了 HybMEC 脏页面同步模块，即源和目的宿主机 QEMU 进程中的缺页响应控制器和缺页请求控制器. 此外，我们还主要修改了 KVM 内核驱动中的内存虚拟化模块 KVM_MMU. 修改后的 KVM_MMU 通过 ioctl 向用户态 QEMU 提供设置 VM 内存页的访问控制的接口. 同时，KVM_MMU 还将捕获 VM 对脏页面访问，并通过 dpage_pull 回调函数向 QEMU 提交网络缺页处理请求.

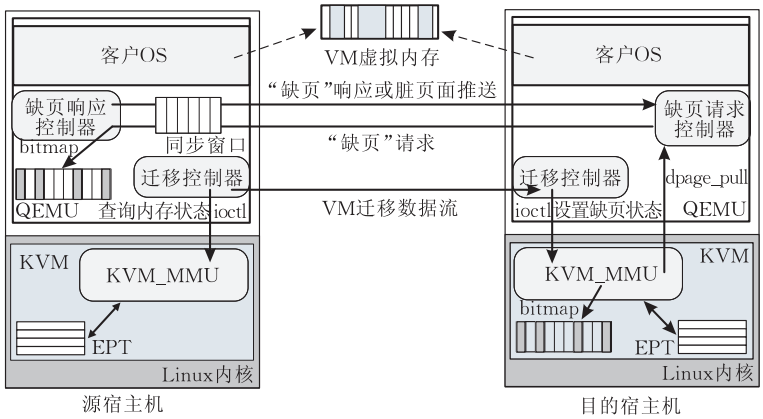

图 3 HybMEC 原型实现

3.1 迁移数据流处理

在迁移过程中，VM 内存、VCPU 寄存器、内存以及其它虚拟设备状态数据以迁移数据流的形式从源宿主机传输到目的宿主机. 为了保证 VM 状态同步的正确性，HybMEC 需要准确控制数据流在源和目的宿主机上的发送和接收. 同时，这也要求 HybMEC 明确定义不同虚拟设备在数据流中的格式，并正确区分同一虚拟设备在不同迁移阶段中的数据内容.

如图 4 所示，HybMEC 定义了 4 种类型的数据段格式，其中 WPAGES、BITMAP 和 DPAGE 3 种数据段格式分别对应 3 个内存同步阶段中内存状态数据，剩余的 DEVICE 类型数据段格式则对应 VM 虚拟设备状态数据. 其中，WPAGES 类型数据段用于全内存同步；BITMAP 和 DEVICE 类型数据段则用于内存位图同步；DPAGE 类型数据段则用于脏内存同步. 每一种数据段由首字节标识其类型，后跟一段不定长的设备标识字段指明该数据段所对应的 VM 虚拟设备；设备标识字段后跟随真正的状态数据. 其中，内存状态数据以页为单位，每个页面数据 page_data 的前 8 字节被用于记录该页面信息标记 page_flag：包括该页面首地址(客户 OS 物理地址)以及是否为空闲页等重要信息. 如果 page_flag 指示一个内存页面为空闲页，那么 page_data 字段长度为 1 字节，并且其值等于零；反之，page_data 字段包含了该页面完整的数据内容. 值得注意的是，DEVICE 类型数据段包含了除内存外的所有虚拟设备状态数据，不同设备状态数据之间以虚拟设备标识字段区分.

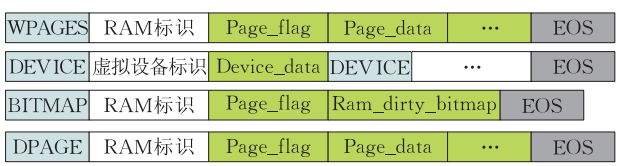

图 4 VM 状态数据流

HybMEC 原型系统中，源、目的宿主机上迁移控制器与缺页控制器分阶段共享数据流通道(socket 连接)用于传输数据流. 其中，迁移控制器负责前两个迁移阶段的数据流传输，而脏页面同步阶段数据流由源、目的宿主机上的缺页控制器控制.

3.2　脏页面按需复制

目的宿主机上 HybMEC 除对脏页面设置缺页状态之外，还需要在内核态 KVM 模块中维护 dirty_bitmap 数据结构. 这样做的直接好处是，避免了在 KVM_MMU 确认当前缺页异常页是否是脏页面时，从内核态切换至用户态查询 dirty_bitmap. 但这也要求 HybMEC 在每一次收到到脏页面数据后，主动地更新内核态中的 dirty_bitmap. 因此，目的宿主机迁移控制器接收到 dirty_bitmap 数据后，HybMEC 通过 ioctl 系统调用访问 KVM_MMU 接口，对所有脏页面设置缺页状态，并在 KVM 创建一个 dirty_bitmap 的复本. 由于 HybMEC 执行上述操作时，VM 在目的宿主机上仍处于暂停状态，因此 KVM_MMU 还未为一些页面建立有效的 EPT 表项. 因此，HybMEC 根据脏页面地址查找对应的 EPT 表项的过程中，还需要适时地为该地址建立各级 EPT 表项，并将最低级 EPT 表项设置 non-present 属性.

当 VM 访问一个未同步的脏页面时，将产生缺页异常而陷入 KVM 异常处理入口函数 handle_exceptoin. 后者根据异常类型，确定是缺页异常后则调用缺页异常处理函数 kvm_mmu_page_fault. kvm_mmu_page_fault 将首先确认产生该缺页异常的原因是否是 VM 对未同步的脏页面的访问，并且当前缺页页面不是空闲页面. 如果上述确认通过，kvm_mmu_page_fault 则调用 dpage_pull 函数向用户态 QEMU 进程转发脏页面缺页事件. QEMU 进程中缺页请求控制器模块接收到该缺页事件后，立即执行脏页面的按需复制，并通过数据流通道向源宿主机上的 QEMU 进程提交“缺页”请求. 需要注意的是，如果 kvm_mmu_page_fault 函数确认缺页脏页面为空闲页面，那么该函数直接为该页分配一个零页面，并调用 vm_lanch 恢复 VM 运行. 对于其它情况的缺页异常，kvm_mmu_page_fault 将按正常缺页异常处理逻辑执行.

3.3　脏页面预取

源宿主机上的 QEMU 进程在进入脏页面同步阶段时，缺页响应控制器模块将自动开启同步定时器，并初始化同步窗口. 其中，定时器间隔长短以及同步窗口大小都会对脏页面同步效率以及 VM 应用性能造成影响. 例如，窗口过小或定时间隔过长，可能延长脏页面同步阶段的持续时间，并导致过多的网络缺页处理而造成 VM 应用性能的抖动；过大的窗口或过短的超时间隔，可能会导致脏页面同步过程占用过多的物理资源而造成 VM 应用性能的明显下降. 目前 HybMEC 原型实现中，我们将定时器间隔设定为 10 ms，同步窗口大小等于 64 页(256 KB). 通过定量的实验分析，明确上述两个参数对 HybMEC 迁移性能的影响将是我们下一步工作内容之一.

当源宿主机上同步定时器发生超时时，缺页响应控制器会将同步窗口首地址移置窗口上一次结束位置的下一个脏页面处，并依次将页框号(frame number)大于窗口当前起始页面的剩余脏页面填入同步窗口，直至窗口剩余空间或剩余脏页面数为零. 随后，缺页响应控制器开始推送同步窗口中的脏页面数据，并在本次推送结束时更新本地 dirty_bitmap 数据以及同步定时器，等待同步定时器下一次超时的到来. 如果源宿主机 QEMU 进程接收到缺页请求，那么缺页响应控制器立刻结束当前的定时器等待而进入缺页响应处理. 此时，缺页响应控制器立即将同步窗口首地址调整至当前缺页页面处，并将该页面剩余邻居脏页面填入窗口，直至窗口剩余空间或剩余脏页面数为零；接着，缺页响应控制器立即以窗口中的页面数据响应缺页请求. 此外，在源宿主机上，缺页响应控制器真正传输一个页面数据前同样需要确认该页面是否为空闲页，避免空闲的脏页面通过“捎带”或者定时推送以完整页数据的形式被复制.

在目的宿主机上，QEMU 进程中的缺页请求控制器除了控制向源宿主机 QEMU 进程提交缺页请求，还负责接收脏页面数据流. 在接收到脏页面数据后，缺页请求控制器通过 ioctl 调用通知内核态 KVM_MMU 更新 VM 内存页面映射关系以及 dirty_bitmap 数据.

4　实验评估

前文介绍了 HybMEC 原型系统的实现细节，本节将通过具体实验验证 HybMEC 的有效性和可用性，并说明各项关键技术对于迁移性能参数的影响. 在多种不同类型的应用负载下，我们将对比 KVM Pre-copy 机制，并关注总迁移时间、VM 停机时间、内存同步数据量以及 VM 应用性能损失等性能参数. 最终的实验结果表明，相比于 Pre-copy 机制，HybMEC 不仅减少了迁移数据量、降低了迁移过程造成的 VM 运行性能损失，而且缩短了 VM 停机时间以及总迁移时间. 特别是对具有访存密集型

应用的 VM，HybMEC 表现出明显的稳定性和高效性.

4.1 实验环境

我们在 DELL T1500 工作站上完成所有迁移实验.各物理机配置：Intel(R) Core(TM) i7 860@2.80GHz CPU，8GB 内存，320GB SATA 硬盘以及 Broadcom BCM57780 Gigabit 网卡.物理机之间通过 H3C S5120-28C-EI 千兆交换机连接；所有的 VM 磁盘镜像文件存放在 iSCSI 共享存储服务器.KVM 宿主机软件环境为 Ubutun Server 10.04 TLS，内核版本 2.6.32-24，KVM 版本 kvm-88.实验中，VM 均配置了一个 VCPU，并安装 Ubuntu Desktop 10.04 LTS 版本操作系统，内存大小介于 256 MB～2 GB 之间.此外，所有 VM 通过桥接模式接入物理网络.

如果没有特别指出，实验中 VM 虚拟内存大小默认设置为 512 MB.另外，同一组应用负载下 VM 迁移实验被重复 10 次，本节中实验结果为 10 次结果均值.

4.2 内存压力负载

为了验证访存密集型负载条件下，HybMEC 迁移的高效性和可用性，我们以一个内部持续运行内存更新程序的 VM 作为迁移对象.其中，内存更新程序模拟一个访存密集型应用，以页面为单位，按照指定的速率循环更新其线性地址空间内的一块内存区域；每次访问一个内存页时，该程序将向内存页首字节写入一个随机数.每一组迁移实验中，当内存更新程序运行 15 s 后，VM 开始执行在线迁移

如图 5 所示，当内存更新速率由 0 逐渐上升至 10240 页/s，Pre-copy 机制迁移性能急剧下降.其中，总迁移时间从最初的 8.3901 s 上升至 188.7537 s.实验中，我们发现当内存更新速率大于 16348 页/s，Pre-copy 迁移即出现无法正常结束的情况.因此，Pre-copy 机制的内存压力负载实验中，我们只给出了内存更新速率小于 10240 页/s 的实验结果.最后，我们还发现即使在相同内存负载条件下，Pre-copy 机制的总迁移时间具有明显的不稳定性.例如，当内存更新速率设定为 10240 页/s，总迁移时间的最大差值接近 75 s.

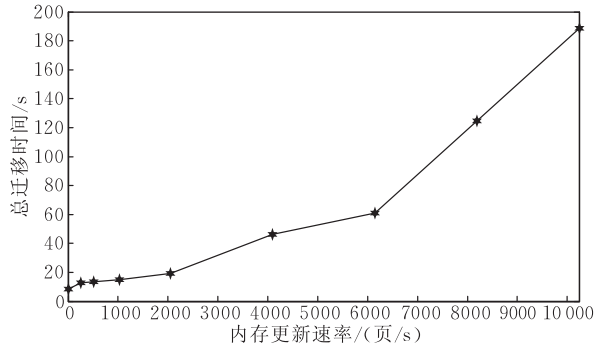

图 5 Pre-copy 迁移总时间随内存更新速率的变化

上述的实验结果表明，VM 内存更新速率明显地制约着 Pre-copy 机制迁移性能.随着内存更新速率上升，迁移性能出现大幅度下降同时还伴随着强烈的不稳定性.通过分析 KVM（版本 kvm-86～kvm-0.12.4）的 Pre-copy 机制源代码，我们发现造成上述结果的原因在于 KVM 仅以最小脏内存数作为内存迭代同步的终止条件，而未考虑通过设置最大迭代次数或最大数据传输量限制迭代同步.因此，当 VM 内存更新速率大于物理网络传输速率时，Pre-copy 机制因为剩余脏页面数不能收敛于最小脏内存数而导致迁移失败.但是，由于最小剩余脏内存数的限制，Pre-copy 机制在不同的负载下停机时间内复制的数据量大致相等，这使得 VM 停机时间反而不随着内存更新速率的上升而表现出明显变化，如图 6 所示.但是，这样的稳定性以剩余脏内存数可收敛为前提，以延长总迁移时间为代价.

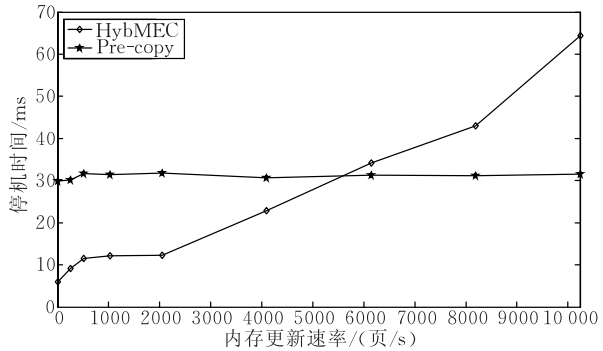

图 6 HybMEC 与 Pre-copy 迁移停机时间随内存更新速率的变化

虽然，不同的虚拟化平台对于 Pre-copy 机制在实现细节上不尽相同，但是通过上述的实验及分析，我们有理由相信由于依赖于剩余脏内存收敛速率，Pre-copy 机制在访存密集型负载下不能保证理想的迁移性能.相反，因为不依赖于迭代同步，所以即使 VM 内存更新速率远大于物理网络传输速率，HybMEC 亦能高效地完成迁移任务.如图 7 所示，当 VM 内存更新速率由 0 增加至 65536 页/s，总迁移时间只是缓慢增加至最初值的 10.2 倍.即便是内存更新速率上升为 65536 页/s，HybMEC 迁移平均持续时间为 32 s，平均复制了 203100 个内存页面，其中大约有 67930 个页面（约 265 MB）经过脏内存同步的二次复制.图 6 对比了两种迁移机制的 VM

停机时间随内存更新速率的上升的变化情况. 我们发现，当内存更新速率较小的情况下，HybMEC 的 VM 停机时间明显低于 Pre-copy 机制. 这是因为在停机时间内，HybMEC 只需要复制脏页面位图数据，并且脏页面缺页状态设置执行延时小于 Pre-copy 的剩余脏内存复制延时，因此 HybMEC 能够保证相对更小的 VM 停机时间. 但是，随着内存更新速率的进一步上升，停机时刻剩余脏页面数增加，HybMEC 执行脏页面缺页状态设置的延时增大. 因此，当内存更新速率大致为 5600 页/s(约 22MB/s)时，HybMEC 的 VM 停机时间超过 Pre-copy 机制. 然而，实验中我们也发现即便内存更新速率上升至 65536 页/s，HybMEC 也只造成 293.024 ms 的停机时间. 在图 8 中，我们给出了相同内存更新速率的条件下，两种机制在迁移过程中内存同步数据量对比情况. 从图中我们看到，随着内存更新速率的增大，两种机制在同步数据量上的差距被迅速拉大. 其原因在于随着内存更新速率的增大，迁移过程中 Pre-copy 机制重复复制的内存页面数急剧上升，而 HybMEC 只有部分的脏页面执行额外的一次复制. 此外，HybMEC 对于空闲页面的处理进一步减小了被复制的内存页数量.

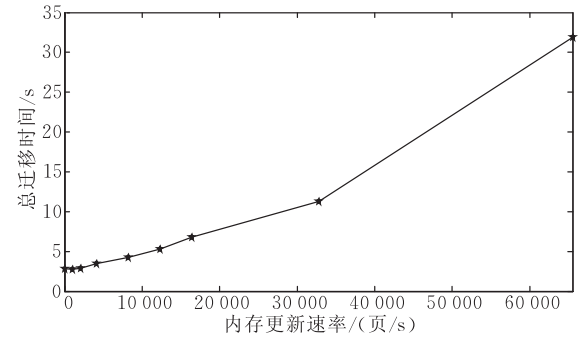

图 7　HybMEC 迁移总时间随内存更新速率的变化

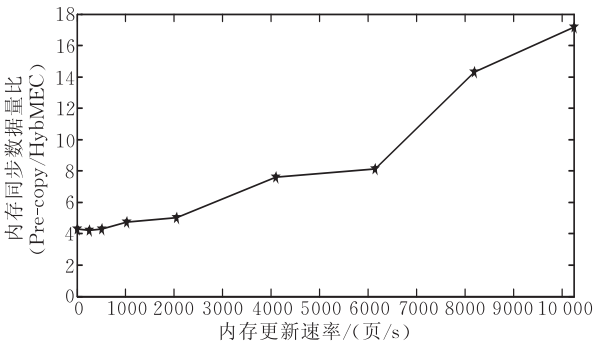

图 8　Pre-copy 与 HybMEC 迁移内存同步数据量对比

上述 3 方面的迁移性能对比结果说明，由于依赖于剩余脏内存收敛速率，Pre-copy 机制在访存密集型应用负载下，并不能保证理想的迁移性能，甚至出现迁移失败的情况. 然而，HybMEC 通过推送和按需复制相结合的方式，减缓了内存更新对迁移性能的影响，加速迁移过程的同时，大幅度地降低同步数据量，减少了迁移过程对物理资源的消耗.

4.3　并发 Web 负载

为了研究 HybMEC 在迁移过程中对 VM 应用 I/O 性能造成的影响，我们以内部运行 Apache2 服务器的 VM 为迁移对象，跟踪并分析了迁移过程中 Apache2 服务器在并发访问下输出带宽的变化情况. 实验中，我们在第 3 台物理机上模拟了 50 个并发客户端持续从 Apache2 服务器下载 512 KB 静态文件的行为.

图 9 和图 10 分别给出了 Pre-copy 和 HybMEC 迁移过程 Apache2 服务器输出带宽的变化曲线. 其中，两次迁移实验分别从图中的 10.5 s 和 1.5 s 时刻开始. 图中我们可以看出，两种迁移实验中输出带宽曲线均出现了瞬时下降和恢复. 但是，HybMEC 曲线中 Apache2 输出带宽出现明显波动，即从最小带宽值恢复至正常波动范围的持续时间要小于 Pre-copy 机制. 出现上述实验结果的原因在于 HybMEC 在目的宿主机上恢复 VM 执行后，通过“定时推送”以及“预取页”加速了脏内存页数据的同步过程，避免了频繁的脏页面按需复制对 VM 运行性能的影响. 此外，将 VM 切换到目的宿主机上运行之前，HybMEC 通过为脏页面设置缺页状态的操作，预先为脏页面在 EPT 表中建立了地址映射关系. 这样，当 VM 访问上述脏页面时不再因为地址映射关系未建

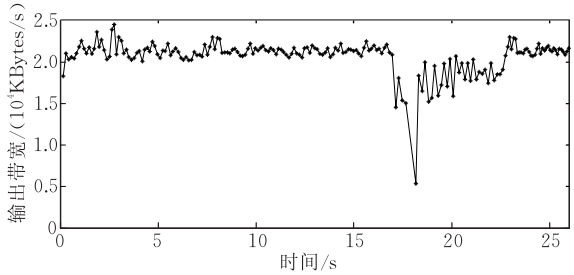

图 9　Pre-copy 迁移过程中 Apache2 输出带宽变化

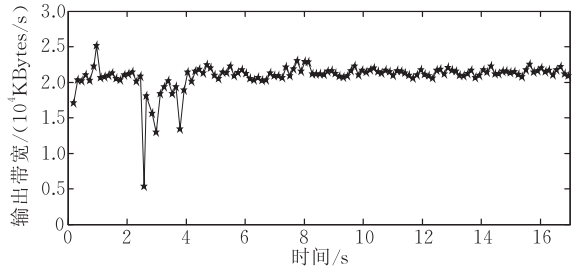

图 10　HybMEC 迁移过程中 Apache2 输出带宽变化

立而通过缺页异常陷入内核态 KVM_MMU 处理地址映射关系的建立. 相反,Pre-copy 机制由于在切换 VM 运行的时刻,目的宿主机上 EPT 基本为空,KVM 需要在后续的运行过程中通过缺页异常处理动态地为 VM 访问的页面建立地址映射关系. 因此,如图 9 所示,Pre-copy 机制造成 VM 性能在迁移结束后的 5 s 时间里才恢复正常. 此外,在实验中我们也发现与内存压力负载实验类似的结果. 在并发 Web 负载下,HybMEC 的迁移持续时间仍明显小于 Pre-copy 机制. 上述实验结果说明,即使是高并发 I/O 负载条件下,HybMEC 在脏内存同步阶段,采用定时推送复制辅助按需复制的方式,保证了 VM 应用在迁移过程中更为稳定的 I/O 性能.

4.4 综合负载对比

实验的最后,我们对比了 4 种不同类型负载下 HybMEC 与 KVM Pre-copy 机制迁移的性能,包括总迁移时间、VM 停机时间以及内存同步数据量,实验结果分别如图 11～图 13 所示. 实验中,除了上述内存压力负载(Mem_stress)和并发 Web 负载(Apache2_web),我们还选取 Linux 内核编译(Liux-kbuild)以及空闲系统(idle)作为迁移实验的应用负载. 其中,Linux-kbuild 是一种系统调用密集型应用负载,其对于 CPU、内存以及磁盘文件读写的操作需求大致均衡;而 idle 负载除了必要的系统程序随客户 OS 启动后,无额外的应用程序运行,近似为低负载应用场景.

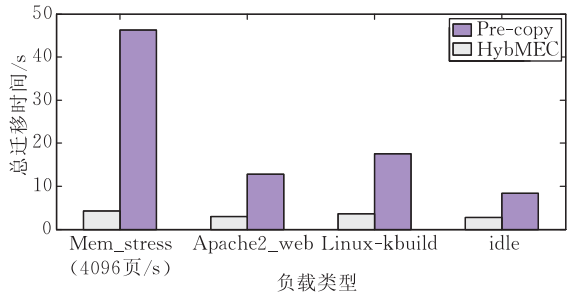

图 11 总迁移时间对比

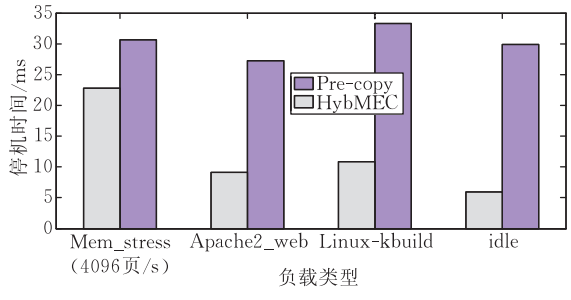

图 12 VM 停机时间对比

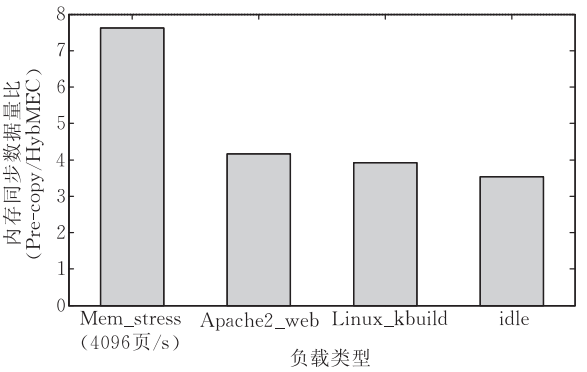

图 13 内存同步数据量比

从图 11 中我们不难看出 4 种负载情况下,相对于 Pre-copy 机制,HybMEC 均大幅地缩短了迁移总时间. 其中,在内存更新速率最大的 Mem_stress 负载下(内存更新速率被设定为 4096 页/s,即 16 MB/s),两种机制的迁移持续时间差距最大. 在该实验中,HybMEC 平均复制了 76684 个内存页面,其中大约有 5200 个页面(约 20.3 MB)经过脏页面同步二次复制;而 Pre-copy 在完成第一轮内存复制后,即近入了迭代同步过程直至剩余脏页面数小于预设值 10,导致总迁移时间依赖于剩余脏页面收敛速率而被延长. 而即使在最理想的 idle 负载实验里,HybMEC 迁移持续时间仍明显小于 Pre-copy 机制. 在图 12 中,我们看到在相同的负载情况下,HybMEC 均取得了更小的停机时间. 其中,在 idle 负载条件下 HybMEC 取得了最小的 VM 停机时间. 其原因如前文所分析的,由于 idle 负载的内存更新速率小于其它负载情况,在停机阶段剩余的脏页面数也少于其它负载. 这也使得 HybMEC 在停机时间内设置脏页面缺页状态的执行延时相对于其它应用负载要小. 相反,Pre-copy 机制在停机时间内需要复制剩余脏内存页数据,并且复制的页面数不大于预设的最小剩余脏页面数. 因此,虽然 Pre-copy 在 4 种负载下的停机时间大致相当,约为 30 ms,但是仍大于 HybMEC 4 种负载实验中的 12 ms 平均时间. 最后,在图 13 中,我们对比两种迁移机制在 4 种类型负载下的内存同步数据量. 即使是理想的 idle 负载,Pre-copy 迁移的内存同步数据量仍然是 HybMEC 的 3.72 倍.

通过上述 4 种不同类型负载下的迁移实验对比,我们不难发现 HybMEC 在减少迁移同步数据量、缩短总迁移时间以及减小 VM 应用性能损失等方面明显好于现有 Pre-copy 机制. 同时,实验结果从另一个侧面验证了 Pre-copy 机制对于访存密集

型负载的低效性和不适用性.同比于其它 3 种类型负载,Pre-copy 机制在内存压力负载的实验中各方面的迁移性能均取得最差结果.对剩余脏内存收敛速率的过分依赖是造成该结果的根本原因.但是,HybMEC 通过内存混合复制方式以及页面数据复制优化,大幅地减小了由于 VM 内存频繁更新所造成的各方面迁移性能下降.同时,HybMEC 在复制页面时对于空闲页的优化处理,大幅度地削减了内存同步数据量,并提升了 VM 状态迁移的整体速度.

5　相关工作

最早由 Clark 等人[5-6]提出并实现的基于内存预拷贝(Pre-copy)方式的 VM 在线迁移机制,将恢复 VM 在目的宿主机上的运行推迟在内存一致性状态同步完成之后.其中,内存状态需要通过多轮的脏内存复制实现同步.目前,Xen[3]、VMware[5]及 KVM[9]等主流系统虚拟化平台上均提供了对于该机制的实现,并广泛应用于局域网环境中跨物理机的 VM 快速迁移.由于需要经过多轮迭代复制才能实现迁移目的端对 VM 内存状态的一致性备份,而机制本身不能控制或预测迁移过程中 VM 内存更新速率,因此 Pre-copy 机制存在内存同步的收敛性问题[15].该问题是指当网络传输速率小于 VM 修改其内存的速率时,内存状态的迭代复制过程由于不满足最少剩余脏页面数而不能主动停止.目前,解决上述问题的普遍方法是通过预设定最大迭代次数或数据传输量,约束迁移过程中内存状态的迭代复制.但是,对于具有大内存或访存密集型应用的 VM,在停机阶段里大量剩余脏内存数据等待复制,延缓了 VM 在目的宿主机上的恢复运行,这势必延长了 VM 停机时间,并影响 VM 在迁移过程中的外部可访问性.类似的,在低速网络环境中执行 VM 在线迁移时,Pre-copy 机制也可能遇到上述的收敛性问题.

另外,Pre-copy 机制的内存迭代复制还造成了迁移数据的高冗余性,导致了迁移过程对物理资源的不合理消耗.最近,在一些研究中提出利用内存压缩[16-17]或消除内存冗余数据[18]等方法减小迁移数据量.但是,这些方法仍以 Pre-copy 迁移机制作为性能优化前提,本质上没有脱离两阶段迁移基本模式,也没有从根本上消除迁移过程对内存迭代同步的依赖.此外,压缩算法的引入势必会造成迁移过程对宿主机 CPU 资源额外的消耗.

基于内存后拷贝(Post-copy)方式的 VM 在线迁移机制[15,19],将内存的同步过程推迟到 VM 在目的宿主机上恢复运行之后.迁移过程首先在两端宿主机之间同步 VM 的 VCPU 以及除内存外的虚拟设备的状态,接着立即在目的宿主机上恢复 VM 执行.随后,VM 内存数据则通过目的宿主机按需取页的方式实现同步.Post-copy 机制中 VM 在目的宿主机上的恢复执行先于内存状态同步,所以在 VM 恢复执行的时刻,只有源宿主机拥有完整的 VM 内存状态.Post-copy 机制能够保证在迁移过程中每一个内存页至多被复制一次.因此,该机制从根本上避免了冗余数据对网络带宽的不必要的占用.然而,VM 切换至目的宿主机上运行,并访问一个未同步的内存页面时,Post-copy 机制被迫中断 VM 正常运行,通过网络缺页处理及时地解决该页面的加载问题.虽然,类似 HybMEC 采用的预取页机制能够帮助 Post-copy 机制减少网络缺页处理的次数,但是,由于驻留在源宿主机上的内存页面数远大于 HybMEC 中的脏页面数,Post-copy 机制仍无法避免频繁的按需加载内存页所造成的 VM 性能下降以及内部应用执行延时的上升.此外,上述工作在实现上未能很好地满足在线迁移机制的透明性要求.Hines 等人[15]的工作需要分别修改客户 OS 以及宿主机 OS 的内核以实现动态内存气泡回收和内存页面的按需加载.因此,该方法中预先正确配置客户 OS 以及宿主机 OS 是 VM 能够执行 Post-copy 迁移的前提.Hirofuchi[19]在工作中,网络缺页处理需要通过除 QEMU 进程以及 KVM 内核模块外的辅助用户态进程和内核模块完成.由于辅助进程在启动时即严格绑定与 VM 内存映射关系,因此在 VM 执行在线迁移的前后,该辅助进程不能由 KVM 控制启动或停止,而需要通过手动控制.

最近,VM 动态克隆系统 SnowFlock[20]利用了与 Post-copy 机制类似的 VM 内存加载方式.当开始执行克隆操作时,原 VM 的 VCPU 及虚拟设备的状态被迅速复制到目标宿主机上;接着,原 VM 和克隆 VM 各自恢复执行,并且分别按照“写时复制”和“按需加载”方式处理内存访问.当克隆 VM 访问未加载的内存页时,VMM 通过网络缺页处理从原 VM 复制页面数据.但是,通过 HybMEC 的工作我们有理由相信,仅依靠“按需加载”处理克隆 VM 对内存页的缺页访问,SnowFlock 很难保证克隆 VM 内部应用的执行性能和效率.此外,SnowFlock 在实现上同样需要修改 Xen 及 Linux 客户 OS 以实现 VMM 对 VM 访存的捕获和内存页面的网络加载.另

外,与 VM 在线迁移工作类似,VM 快照恢复同样需要解决 VM 内存页加载的问题,区别在于后者的内存数据从本地或网络磁盘上的快照文件中加载. Zhang 等人[21]提出利用"懒恢复"(lazily restoring)机制实现对 VM 内存快照的快速加载,即 VM 快照恢复的初始阶段,预先加载内存工作集(working set)中的内存页面. 该工作更多地关注执行 VM 快照过程对内存工作集的跟踪以及内存工作集中页面数据的存储. 与 HybMEC 类似,该工作中利用了内存硬件辅助虚拟化技术,以实现 VM 内存访问跟踪.

6 结论与下一步工作

虚拟机技术充分利用复用单个物理机资源的优势,结合其灵活性、封装性,实现了物理资源的高度共享、资源利用率的显著提升以及软件应用的灵活部署. 而 VM 在线迁移技术更是充分利用 VM 技术的封装性和灵活性,实现了 VM 运行状态通过计算机网络在物理机之间的快速迁移. 通过结合源宿主机复制推送和目的宿主机按需复制两种方式,本文提出了基于内存混合复制方式的 VM 在线迁移机制 HybMEC. 在实现上,HybMEC 不需要修改或重新编译客户机操作系统内核,对客户机操作系统及其应用程序完全透明.

本文的基本思想是:利用源宿主机主动推送的方式完成绝大多数的内存数据的一次性复制;而将少数状态不一致的脏内存数据推迟到 VM 在目的宿主机上恢复执行之后,通过按需复制或定时推送的方式实现同步. 相对于已有的迁移机制,由于不依赖于内存轮迭代同步,HybMEC 能够大幅度地减少迁移过程中的同步数据量,并缩短总迁移时间. 同时,在 VM 停机时间内,HybMEC 不需要复制剩余脏页面数据,从而有效地缩短了 VM 停机造成的不可访问时间. 为了进一步优化迁移性能,本文还对内存页数据的复制做了重要的优化,例如采用预取页机制以减少脏内存按需复制造成的 VM 性能损失,采用定时推送机制以缩短脏内存同步过程的持续时间,以及禁止空闲脏页面引发网络缺页处理以避免 VM 运行状态发生不必要的状态切换.

基于 KVM 上,我们设计并实现了 HybMEC 原型系统,并通过多种不同类型的应用负载验证了 HybMEC 机制的高效性和可用性. 实验结果表明,即使对于访存密集型应用,HybMEC 也能够保证更短的 VM 停机时间以及总迁移时间,缩短迁移过程对源宿主机的长时间依赖. 同时,HybMEC 极大地降低了迁移过程同步数据量,减少了迁移过程本身对物理资源的占用.

当 VM 在目的宿主机上恢复执行后,HybMEC 迁移过程开始进入脏页面同步阶段. 在此阶段中,VM 的完整状态需要由源和目的宿主机共同维护:源宿主机拥有正确的脏内存状态;而目的宿主机拥有 VM 当前 VCPU、部分内存以及其它虚拟设备的状态. 因此,在该阶段中源或目的宿主机的宕机不但会导致 VM 迁移过程失败,而且还会造成 VM 运行状态的丢失. 因此,HybMEC 如何保证上述阶段中的迁移可靠性将成为我们下一步即将开展的重要工作.

参 考 文 献

[1] Nurmi D, Wolski R et al. Eucalyptus: A technical report on an elastic utility computing architecture linking your programs to useful systems. UCSB Computer Science, Santa Barbara, California: Technical Report 2008-10, 2008

[2] Armbrust M, Fox A, Griffith R et al. Above the clouds: A Berkeley view of cloud computing. UC Berkeley, EECS: Technical Report EECS-2009-28, 2009

[3] Barham P, Dragovic B, Fraser K et al. Xen and the art of virtualization//Proceedings of the 19th ACM Symposium on Operating Systems Principles. Bolton Landing, NY, USA: ACM, 2003: 164-177

[4] Goldberg R P. Survey of virtual machine research. IEEE Computer, 1974, 7(6): 34-45

[5] Nelson M, Lim B-H, Hutchins G. Fast transparent migration for virtual machines//Proceedings of the Annual Conference on USENIX Annual Technical Conference. Anaheim, CA: USENIX Association, 2005: 25-25

[6] Clark C, Fraser K, Hand S et al. Live migration of virtual machines//Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2: USENIX Association. Berkeley, CA, USA, 2005: 273-286

[7] Zayas E. Attacking the process migration bottleneck. SIGOPS Operating Systems Review, 1987, 21(5): 13-24

[8] Noack M. Comparative evaluation of process migration algorithms[M. S. dissertation]. Dresden University of Technology-Operating Systems Group, German, 2003

[9] Kivity A, Kamay Y, Laor D et al. KVM: The Linux virtual machine monitor//Proceedings of the Linux Symposium (OLS'07). Ottawa, Ontario, Canada, 2007: 225-230

[10] Tanenbaum A S. Modern Operating Systems. 3rd Edition. Upper Saddle River, New Jersey 07458: Prentice Hall, 2007

[11] Denning P J. The working set model for program behavior. Communications of the ACM, 1968, 11(5): 323-333

[12] Waldspurger C A. Memory resource management in VMware ESX server. SIGOPS Operating Systems Review, 2002, 36 (Sup. I): 181-194

[13] Intel. Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture, 2011: 35-104

[14] AMD. AMD64 Architecture Programmer's Manual Volume 2: System Programming, 2006: 1-124

[15] Hines M R, Deshpande U, Gopalan K. Post-copy live migration of virtual machines. SIGOPS Operating Systems Review, 2009, 43(3): 14-26

[16] Svard P, Hudzia B, Tordsson J et al. Evaluation of delta compression techniques for efficient live migration of large virtual machines//Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Newport Beach, California, USA: ACM, 2011: 111-120

[17] Jin H, Deng L, Wu S et al. Live virtual machine migration with adaptive, memory compression//Proceedings of the 2009 IEEE International Conference on Cluster Computing. New Orleans, Louisiana, USA, 2009: 1-10

[18] Xiang Z, Zhigang H, Jie M et al. Exploiting data deduplication to accelerate live virtual machine migration//Proceedings of the 2010 IEEE International Conference on Cluster Computing. Heraklion, Crete, 2010: 88-96

[19] Hirofuchi T, Nakada H, Itoh S et al. Enabling instantaneous relocation of virtual machines with a lightweight VMM extension//Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing. Melbourne, Australia, 2010: 73-83

[20] Lagar-Cavilla H A, Whitney J A, Bryant R et al. SnowFlock: Virtual machine cloning as a first-class cloud primitive. ACM Transactions on Computer Systems, 2011, 29(1): 1-45

[21] Zhang I, Garthwaite A, Baskakov Y et al. Fast restore of checkpointed memory using working set estimation//Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Newport Beach, California, USA: ACM, 2011: 87-98

CHEN Yang, born in 1983, Ph. D. candidate. His main research interests include virtualization technology, distributed computing.

HUAI Jin-Peng, born in 1962, professor, Ph. D. supervisor. His main research interests include grid computing, distributed computing, network security.

HU Chun-Ming, born in 1977, Ph. D., associate professor. His main research interests include virtualization technology, grid computing, distributed computing.

Background

One attractive functionality of system virtualization technology is the ability to migrate a VM from one physical machine to another on-the-fly. It enables migrating an operating system instance across difference hosts with little service interruption. Live migration is widely adopted in large-scale virtual machine environments, such as virtual data centers, to support dynamic workload balancing, scheduled machine maintenance, and power management. Greater efforts have been taken towards mechanisms for efficient live virtual migration, e. g., pre-copy based and post-copy based approaches. However, exiting mechanisms have to be in the face of some problem, such as convergence of iteration, redundancy of copying, and guest OS transparence. The critical issue of live migration is efficiently replicating the VM memory state over the physical network between the two physical hosts. Pre-copy approach has performance and efficiency restrictions due to its inherently dependent of iterated memory state replication. Although, some related work has been taken to optimize the memory state replication, such as memory compression and redundancy elimination, its improvement is limited due to the fetters of iterated replication. While Post-copy approach could sustain great VM performance degradation due to the long latency of networked demanding paging.

This paper proposes an advanced live migration mechanism based on hybrid memory copy approach, called HybMEC. The authors combine active pushing copy with on-demand copy, to accelerate the process of memory state migration. Different with exiting mechanisms, HybMEC completely shakes the memory state iterated replication off. It firstly replicates the whole RAM memory from the source host to the destination host, and then defers the dirty memory copying phase until after the VM has been resumed on the target machine. All the dirty pages are timely replicated from the source to target by means of on-demand copying and scheduled pushing. In addition, the authors leverage both pre-paging and free pages avoidance to reduce the performance degradation of VM during the migration, and make further acceleration for the process of migration. The authors' approach also mitigates the physical resources consumption of the live migration process.

Research areas of the authors include live migration of virtual machine, memory virtualization, dynamic virtual cluster provision, and security issues in system virtualization. Those works are mainly supported by the National Basic Research Program (973 Program) of China under Grant No. 2011CB302602, the National High Technology Research and Development Program (863 Program) under Grant No. 2011AA010500, National Natural Science Funds under Grant Nos. 90818028, 91018004.