面向同驻虚拟机的高效共享内存文件系统

沙行勉[1] 吴 挺[1] 诸葛晴凤[3] 杨朝树[1] 马竹琳[1] 陈咸彰[1),2)]

[1)](重庆大学计算机学院 重庆 400044)

[2)](重庆大学通信工程学院 重庆 400044)

[3)](华东师范大学计算机科学与软件工程学院 上海 200062)

摘 要 云环境中虚拟机之间、虚拟机与宿主机之间存在大量文件传输操作,带来极大性能开销.针对同驻一台物理机的多个虚拟机,共享文件系统是提高文件传输性能的有效途径.新型的非易失性内存具有内存级读写速度、存储密度高、可字节寻址和持久化等优点,可用作高性能的共享文件存储设备.然而,现有共享文件系统的设计基于网路通信或虚拟 I/O,不能充分发挥新型非易失性内存的优势.该文面向同驻虚拟机提出一个新型共享内存文件系统设计,在虚拟机之间、虚拟机与宿主机之间提供高效的文件共享机制.在该设计中,共享文件系统被安装在虚拟机与宿主机共享的模拟非易失性内存中,通过共享的页表组织共享文件的数据页.共享文件系统使用虚拟地址空间和处理器中既有的硬件 MMU 直接访问共享文件,减少文件访问 I/O 的软件层次和数据拷贝产生的性能开销.文中还对共享数据的并发访问和一致性提供高效的同步机制.该文采用提出的设计,在 KVM 平台上实现一个高效的共享内存文件系统 StargateFS.实验测试结果显示:StargateFS 的共享文件平均读写性能比目前最先进的共享文件系统 VirtFS 快 64 倍,比 Samba 和 NFS 分别快 172 倍和 191 倍.

关键词 虚拟机;共享文件系统;内存文件系统;共享页表;同步机制;云计算

中图法分类号 TP302 **DOI 号** 10.11897/SP.J.1016.2019.00800

An Efficient Shared In-Memory File System for Co-Resident Virtual Machines

Edwin H-M SHA[1)] WU Ting[1)] ZHUGE Qing-Feng[3)] YANG Chao-Shu[1)]

MA Zhu-Lin[1)] CHEN Xian-Zhang[1),2)]

[1)](*College of Computer Science, Chongqing University, Chongqing* 400044)

[2)](*College of Communication Engineering, Chongqing University, Chongqing* 400044)

[3)](*School of Computer Science and Software Engineering, East China Normal University, Shanghai* 200062)

Abstract The vast file transfers among virtual machines or between virtual machines and physical machines in cloud environment bring large overhead for the system. Shared file system for the virtual machines on a same physical machine, i.e., co-resident virtual machines, and the underlying physical machine is an effective approach for improving the performance of file transfers. Emerging Non-Volatile Memory (NVM) is a promising high-performance storage for shared files for the merits of NVM, such as near-DRAM speed, high density, byte-addressability, and persistency. However, existing shared file system designs rely on either network devices or virtual I/Os, which cannot fully exploit the benefits of NVM. This paper proposes a new design of shared in-memory file system that supports efficient file access on the shared files of co-resident virtual machines and the

收稿日期:2017-10-23;在线出版日期:2018-05-17.本课题得到国家"八六三"高技术研究发展计划项目(2015AA015304)、国家自然科学基金(61472052)、中国博士后科学基金(2017M620412)资助.**沙行勉**,男,1964 年生,博士,教授,博士生导师,中国计算机学会(CCF)高级会员,国家千人计划特聘专家,长江学者讲座教授,主要研究领域为新型内存系统、云计算、操作系统、嵌入式系统和软件、通信安全.E-mail:edwinsha@gmail.com.**吴 挺**(通信作者),男,1991 年生,博士研究生,主要研究方向为系统虚拟化、内存文件系统.E-mail:tingwu@cqu.edu.cn.**诸葛晴凤**,女,1970 年生,博士,教授,博士生导师,中国计算机学会(CCF)会员,主要研究领域为操作系统、嵌入式系统和软件、优化算法.**杨朝树**,男,1984 年生,博士研究生,主要研究方向为持久化内存文件系统.**马竹琳**,女,1993 年生,博士研究生,主要研究方向为内存数据库.**陈咸彰**(通信作者),男,1989 年生,博士,讲师,中国计算机学会(CCF)会员,主要研究方向为新型内存系统、文件系统、嵌入式系统和软件、云计算.E-mail:xzchen@cqu.edu.cn.

underlying physical machine. First, the proposed design sets up a dedicated NVM zone on the physical machine as the sharing storage for the shared in-memory file system. The files of the shared in-memory file system are organized in a "shared page table" that is in the same form of the page table of the processes. A file on the shared NVM can be exposed to the contiguous virtual address spaces of multiple co-resident virtual machines and the physical machine by embedding the shared page table to the page table of the processes of the co-resident virtual machines or the physical machine in open operation. Those processes whether running in virtual machines or the underlying physical machine can access the shared file data with high throughput via the continuous virtual addresses. The traversal of the file index, i.e., the shared page table, for locating the shared file data can fully take advantage of the hardware MMU in processor. Different from the approaches of sharing file using network and the virtual I/O, the proposed design highly reduces the overhead of I/O software stacks and redundant data copies by directly accessing files using virtual address space and hardware MMU. Second, the Virtual File System (VFS) caches of the operating systems of co-resident virtual machines may cause inconsistency of shared files for each VFS cache maintains a copy of the sharing file metadata. We solve the inconsistency problem by proactively synchronizing the metadata of the sharing files in each read/write operation. Third, to ensure the concurrency of file accesses, we present a new synchronous mechanism for shared files. The variable of the shared semaphore is stored on the shared NVM of the co-resident virtual machines and the underlying physical machine. The atomicity of the updating operation of the semaphore variable is maintained by the atomic instructions supplied by the primitives of CPUs. Based on the proposed design, we implement an efficient shared in-memory file system StargateFS in the KVM platform in the Linux kernel. We conduct extensive experiments to evaluate the performance of StargateFS. The experimental results show that the average throughput of StargateFS achieves 64 times, 172 times and 191 times higher than VirtFS, the state-of-the-art shared file system, Samba, and NFS, respectively. Besides, the performance of the shared semaphore on StargateFS is 2.3 times that of the semaphore on Nahanni, the state-of-the-art shared memory mechanism of co-resident virtual machines.

Keywords　virtual machine; shared file system; in-memory file system; shared page table; synchronization mechanism; cloud computing

1　引　言

硬件虚拟化技术[1]允许多个操作系统并发地运行在一台物理机上，共享地使用物理机的硬件资源. 例如，在个人计算机中，Linux 用户可以在虚拟机里运行 Windows 系统及其应用程序，而不需要在两个系统之间切换. 在服务器中，硬件虚拟化技术允许不同版本的 Linux 系统同时运行在物理服务器上，以满足用户对不同 Linux 版本的需求[2]. 在高性能计算领域中，把需要特殊函数库的应用程序封装在一台虚拟机中，能够有效地简化应用程序的部署流程[3]. 硬件虚拟化技术是云计算的核心技术之一[4]，虚拟机对操作系统、函数库、应用程序的封装可以减少在不同硬件平台和操作系统上运行软件的开销. 因此，云计算平台中常以虚拟机作为资源分配的单元[5].

虽然把应用程序和服务封装在虚拟机中有许多优点，但是虚拟机的隔离性使得同驻虚拟机之间以及虚拟机与宿主机之间共享文件和数据有很大的性能开销. 在虚拟机中访问宿主机或其他虚拟机中的文件时，有两种方式实现：(1) 把该文件拷贝到虚拟机中；(2) 在宿主机或其他虚拟中搭建文件服务器，虚拟机作为客户端访问文件. 在这两种实现方式中，都需要在虚拟机间、虚拟机宿主机间拷贝文件. 在部分应用场景中，文件仅需在虚拟机间、虚拟机宿主机间拷贝一次或几次，如应用程序安装镜像文件. 然而在诸多大数据应用场景中，虚拟机之间、虚拟机与宿

主机之间需要频繁地传输文件[6-9].

此外,在同驻虚拟机间、虚拟机宿主机间共享文件需要经过多个软件层次,造成性能损失.传统的共享文件系统[10]如 SMB/CIFS[11](Samba① 中实现)、NFS[12]可以用于同驻虚拟机间、虚拟机宿主机间共享文件.但是,传统的共享文件系统没有针对虚拟化平台的专门设计与优化.现有许多研究工作专注于提高同驻虚拟机间及虚拟机与宿主机之间文件共享的性能.以虚拟磁盘[13]形式存在的虚拟化存储可以同时挂载到多个虚拟机实现文件共享,其性能比传统共享文件系统高,但虚拟磁盘中的数据只有在虚拟磁盘为只读模式时才能实现多虚拟机并发访问.半虚拟化文件系统 VirtFS[14]向虚拟机提供系统层服务接口,实现同驻虚拟机之间、虚拟机与宿主机之间文件共享,相比虚拟化存储的性能更高.但 VirtFS 使用 VirtIO[15]环形缓冲区进行消息传递和数据传输存在多次内存拷贝.当使用以上文件共享方法共享内存文件系统时,很大程度限制了文件读写性能.

新型非易失性内存(Non-Volatile Memory,NVM)[16-17]有许多的优点:(1)与 DRAM 相当的访问速度;(2)存储密度高;(3)能耗低;(4)具有非易失;(5)可字节寻址;(6)可直接挂载到系统总线.随着 3D XPoint② 等新型非易失性内存的发展,出现一类新型持久化内存文件系统,如 SIMFS[18]、NOVA[19]、HiNFS[20].在虚拟机同驻的环境下,可以利用新型内存文件系统的优点提高共享文件的访问性能.内存文件系统直接把文件元数据和文件数据存放在内存中,具有如下优点[21]:(1)突破传统磁盘的 I/O 瓶颈,具备极高的文件读写性能;(2)可按字节寻址粒度访问文件数据;(3)可以把存放文件数据的物理页直接映射到进程的虚拟地址空间,使用进程的虚拟地址访问文件数据.如 SIMFS 通过"文件虚拟地址空间"和进程页表结构,利用硬件 MMU(Memory Management Unit)加速访问文件,能够达到接近内存总线带宽的文件数据读写性能.

针对以上问题和机遇,本文面向同驻虚拟机提出了一个新型的共享内存文件系统设计,以支持同驻虚拟机之间、虚拟机与宿主机之间高效地共享文件.根据该设计,本文基于 KVM(Kernel-based Virtual Machine)平台在 Linux 系统中实现了一个高效的共享内存文件系统 StargateFS. StargateFS 安装在宿主机与虚拟机共享的模拟非易失性内存中.StargateFS利用新型非易失性内存可以直接挂载到内存总线和可字节寻址的特点,使用共享文件页表,把共享文件的数据物理页直接映射到虚拟机及宿主机预留的文件虚拟地址空间,通过虚拟地址和既有的硬件 MMU,使用 CPU 的 load/store 指令直接访问共享文件数据,实现虚拟机之间、虚拟机宿主机之间高效的文件共享.

在共享内存文件系统中,文件系统信息、索引节点信息等元数据信息是临界资源,StargateFS 使用共享自旋锁机制实现不同虚拟机、宿主机内核控制路径间同步地访问共享元数据. Linux 系统 VFS(Virtual File System)提供的目录项、索引节点、数据块缓存机制,可能会导致虚拟机 VFS 缓存信息、宿主机 VFS 缓存信息与共享内存文件系统中的信息不一致,StargateFS 提出了如何避免不同虚拟机、宿主机 VFS 缓存间信息不一致的方法.针对常见的应用场景如消费者生产者,StargateFS 提供了共享信号量支持不同虚拟机、宿主机的多个用户进程间同步地访问同一个共享文件.

本文使用标准测试工具 FIO(Flexible I/O tester)③测试 StargateFS 单线程及多线程不同数据块大小的文件读写性能.测试结果表明,文件顺序读写、随机读写性能都有很大的提升,StargateFS 平均读写性能比目前性能最优的共享文件系统 VirtFS快 64 倍,较常使用的 Samba、NFS 分别提高了 172 倍、191 倍.

本文的主要贡献包括以下几个方面:

(1)面向同驻虚拟机提出了一种基于文件虚拟地址空间的共享内存文件系统设计,支持同驻虚拟机之间、虚拟机与宿主机之间高效地共享文件;

(2)基于 KVM 平台,设计并在 Linux 内核中实现了一个高效的共享内存文件系统 StargateFS;

(3)基于虚拟机宿主机间的共享内存设计并实现共享信号量,为不同系统用户进程间同时访问同一共享文件提供一种快速有效的同步机制;

(4)对比测试 StargateFS 和其他共享文件系统在共享宿主机内存文件系统 EXT4-DAX④ 时的性能,实验结果表明 StargateFS 比目前性能最优的

① Samba. https://www.samba.org/samba/docs/SambaIntro.html, 2011.11.27

② Intel and Micron Produce Breakthrough Memory Technology [Online]. https://newsroom.intel.com/news-releases/intel-and-micron-produce-breakthrough-memory-technology/, 2015.7.28

③ Flexible I/O tester [Online]. https://github.com/axboe/fio, 2005.12.4

④ Add support for NV-DIMMs to ext4 [Online]. https://lwn.net/Articles/613384/, 2014.9.25

VirtFS 快 64 倍.

本文第 2 节介绍系统设计的动机和原则；第 3 节介绍共享内存文件系统的设计与实现；第 4 节介绍不同系统用户进程间的同步；第 5 节比较各种文件共享文件系统的性能并分析性能差异；第 6 节介绍相关工作；第 7 节进行总结.

2　设计动机与原则

2.1　设计动机

在设计面向同驻虚拟机的共享内存文件系统前，首先分析使用传统共享文件系统共享内存文件时可能存在的问题. 共享内存文件系统的传统实现方式有两种：

(1) 宿主机上安装内存文件系统，利用传统的通用网络共享文件系统如 Samba、NFS，将其挂载到虚拟机中；

(2) 针对虚拟机同驻的环境提供半虚拟化的文件系统接口，如 VirtFS[14]，来访问宿主机上的内存文件系统.

以上两种方式各有不足，以典型的内存文件系统 EXT4-DAX 为基础，NFS 和 VirtFS 为代表，分别介绍这两种方式. 如图 1 所示，NFS 和 VirtFS 都是基于客户端服务器端的模式，把宿主机的目录共享挂载到虚拟机中. 另外，使用 VirtFS 在虚拟机间共享文件需要宿主机的支持.

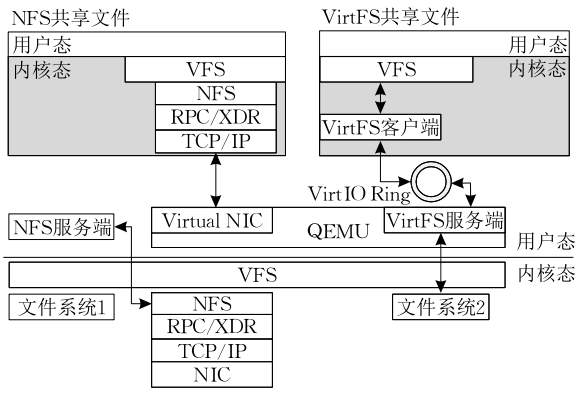


图 1　NFS、VirtFS 结构图

使用 NFS 在虚拟机之间、虚拟机与宿主机之间共享文件时，客户端和服务端通过 RPC(Remote Procedure Call)机制进行通信. 当客户端访问服务端上的内存文件时，NFS 客户端创建 RPC 任务，并向 NFS 服务端发送 RPC 请求，服务端将文件数据封装到应答消息中返回给客户端. NFS 是设计用于独立的物理机之间共享文件，没有为虚拟化环境做专门的优化. 所以即使是位于同一物理机上的多个虚拟机之间、虚拟机与宿主机之间共享文件，NFS 也需要经过诸多面向网络的软件层次，造成极大性能开销. 另一方面，虚拟机使用的是模拟网络设备，多台同驻虚拟机实际是竞争宿主机的单套网络设备，所以存在多层软件模拟竞争的开销，其性能相对于真实网络设备更差[22].

使用 VirtFS 在虚拟机之间、虚拟机与宿主机之间共享文件是通过 VirtIO[15] 中的环形缓冲区传输数据. 在虚拟机中访问共享文件时，VirtFS 服务端把文件数据写入缓冲区，VirtFS 客户端从缓冲区中读取文件数据. VirtFS 不需要把虚拟机对文件系统的操作转换成对虚拟设备驱动的操作，也不需要把对虚拟设备驱动的操作转换成对宿主机文件系统的操作. VirtFS 共享文件的性能相比 NFS 有一定程度的提升，但 VirtFS 只能按照 VirtIO 规定的格式进行数据传输，并且文件数据多次拷入拷出缓冲区会产生极大的性能开销.

总而言之，以上两类共享文件系统均不能充分发挥宿主机上的内存文件系统优势，共享方法的性能瓶颈限制了文件的读写性能. 通过以上分析，可利用半虚拟化提高共享文件接口层次的思想，结合内存文件系统的优点，减少同驻虚拟机之间、虚拟机宿主机之间共享文件的 I/O 软件层次，提高共享文件的访问性能.

2.2　设计原则

StargateFS 遵循以下设计原则：

(1) 实现内存共享. 在虚拟机和宿主机之间需要共享一段物理内存空间，用于安装共享内存文件系统. 为使虚拟机和宿主机内存寻址的差异对共享内存文件系统透明，共享内存中虚拟机对应的物理地址 GPA(Guest Physical Address)与宿主机对应的物理地址 HPA(Host Physical Address)需要一一对应，且在虚拟机和宿主机上都能够通过文件虚拟地址空间正确地访问文件数据物理页. 共享内存是虚拟机和宿主机分别新增的一个内存管理区，需要提供共享内存的专有管理机制；

(2) 缩短共享 I/O 软件栈. 共享内存对宿主机和虚拟机而言就是本地物理内存空间，不需要在虚拟机和宿主机之间传输文件请求，有效地避免共享 I/O 软件栈性能的开销. 共享内存文件系统应避免传统客户端/服务端共享文件系统共享文件经过的网络、VirtIO 中间软件层；

(3) 减少文件数据拷贝. 共享内存文件系统安

装在共享内存中，避免在虚拟机之间、虚拟机宿主机之间拷贝共享文件数据. 共享内存文件系统中的文件默认以 O_DIRECT 模式打开，避免读写文件数据时在 VFS 页缓存(Page cache)上的拷贝；

(4)加速文件数据索引. 利用内存文件系统可把存放文件数据的物理页直接映射到进程的虚拟地址空间的优点，实现 MMU 加速索引文件数据. 共享内存文件系统需要提供一种通用的文件数据索引结构，使得在虚拟机和宿主机系统中，都能通过文件虚拟地址空间利用 MMU 加速访问文件数据；

(5)控制元数据的一致性. 虚拟机和宿主机的 VFS 提供缓存机制，缓存文件系统中的信息如文件索引节点信息. 当更新操作提交到 VFS 后，VFS 并没有立即把缓存信息更新到文件系统中. 因此，为了避免虚拟机、宿主机用户从 VFS 缓存中访问脏的文件系统信息，需要提供相应的机制保证虚拟机的 VFS 中的缓存信息与共享文件系统中的信息的一致性；

(6)同步访问共享数据. 多个虚拟机、宿主机的用户进程并发访问共享内存文件系统中的资源如索引节点信息时，可能会导致共享资源的竞争. 共享内存文件系统应提供多机内核同步机制，控制不同虚拟机、宿主机内核控制路径，对共享内存文件系统中的临界资源互斥地访问. 同样需要提供多机用户进程间的同步机制，确保多个虚拟机、宿主机的用户进程间同步地读写相同共享文件.

3 系统设计与实现

3.1 总体设计

共享内存文件系统 StargateFS 安装在虚拟机与宿主机共享的模拟非易失性内存中，由虚拟机端和宿主机端组成，如图 2 所示. 每端都包含文件数据管理子系统、元数据管理子系统和存储空间管理子系统. StargateFS 的存储空间布局由元数据区和数据区组成. 元数据区包括超级块、索引节点区和共享信号量区. 数据区为文件和共享文件页表提供存储空间. 在虚拟机和宿主机上分别挂载 StargateFS 后，虚拟机用户和宿主机用户都能访问共享内存文件系统中的文件.

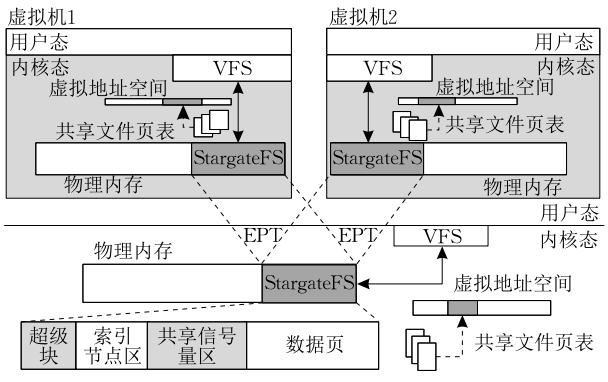


图 2 文件系统总体架构图

在虚拟机和宿主机之间共享文件时，在宿主机上，从预留的内核虚拟地址空间(见 3.4 节)动态映射区中为打开文件分配一段文件虚拟地址空间，把文件的共享文件页表插入到宿主机中内核虚拟地址空间对应的内核页表 PGD(Page Global Directory)级页表项中. 在虚拟机上，从预留的内核虚拟地址空间动态映射区为打开文件分配一段文件虚拟地址空间，把文件的共享文件页表插入到虚拟机中内核虚拟地址空间对应的内核页表的 PGD 级页表项中，如图 3 所示. 最终在虚拟机上和宿主机上都可以通过各自的文件虚拟地址空间访问共享文件数据. 特别地，虚拟机执行文件虚拟地址到文件数据物理地址的转换涉及到两次地址转换，而宿主机只需要经过一次地址转换.

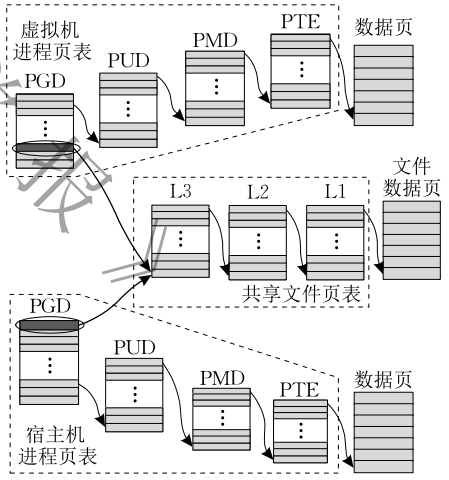


图 3 插入共享页表

在虚拟机之间共享文件时，分别从不同虚拟机预留的内核虚拟地址空间动态映射区中为打开文件分配一段文件虚拟地址空间，把共享文件页表分别插入到不同虚拟机内核虚拟地址空间对应的内核页表 PGD 级页表项中，实现虚拟机间的文件共享. 这种设计能够带来以下好处：

(1)访问速度快. 虚拟机端和宿主机端都可以利用硬件 MMU 加速定位文件数据页.

内存管理单元 MMU 把 CPU 产生的虚拟地址翻译成对应的内存物理地址. 当用户访问文件数据

时，文件系统需要搜索文件索引结构，找到用户请求的文件数据的虚拟地址对应文件的物理地址. 在未使用硬件 MMU 加速的文件系统中，文件索引的搜索是按照软件流程逐级查找文件的索引，会造成较大的性能开销. 而在 StargateFS 中，由于文件数据索引结构与 Linux 系统页表的结构相似，从 MMU 的角度看，两者并无差别. 因此，在虚拟机中访问共享文件时，可以利用硬件 MMU 和虚拟机文件的文件索引将文件数据的虚拟地址，经两次地址转换（虚拟机的虚拟地址到虚拟机的物理地址，再到宿主机的物理地址转换）直接定位到文件数据物理页；在宿主机中访问共享文件时，文件数据的虚拟地址空间经 MMU 一次地址转换直接定位到文件数据物理页. 因此，在 StargateFS 中，虚拟机端和宿主机端都可以利用硬件 MMU 加速定位共享文件数据页.

（2）读写共享文件数据没有缺页错误.

Linux 内存寻址有两种情况会导致 MMU 触发缺页错误：① 当虚拟地址到物理地址没有建立地址映射关系（即页表）时；② 已有映射关系中对应的地址不可被 CPU 直接访问时（例如该地址属于块设备中的交换区）. StargateFS 中不存在这两种导致缺页错误的情况，原因如下：当打开 StargateFS 共享文件时已建立文件虚拟地址空间到共享文件数据物理页之间的映射关系，且该映射关系中的地址都是可以被 CPU 直接访问的非易失性内存物理地址. 当用户进程读共享文件数据时，文件数据所在的位置已通过共享文件页表建好映射关系，读文件数据不会有缺页错误. 当用户进程向共享文件写入数据时，判断文件写入类型，如果是就地更新，与读操作一样，对应位置已经有建好的映射关系；如果是追加写或部分追加写，先申请空闲物理页并更新映射关系（即共享文件页表），所以写共享文件数据也不会产生缺页错误.

（3）在虚拟机和宿主机上都可使用文件虚拟地址空间访问文件数据，通过内存拷贝直接读写共享文件数据.

实现这种设计需要解决以下 3 个关键问题：

（1）虚拟机内存虚拟化引入了一层新的地址空间，虚拟机物理地址空间. 虚拟机的操作系统认为虚拟机物理地址就是实际的物理地址. 然而，虚拟机的物理地址不能直接发送到真实物理机的系统总线，需要 VMM（Virtual Machine Monitor）截获虚拟机对虚拟机物理地址的访问，将其转换成宿主机物理地址. 由此形成了虚拟机虚拟地址 GVA（Guest Virtual Address）到虚拟机物理地址 GPA，再到宿主机物理地址 HPA 的两次地址转换. 当虚拟机内存寻址时，需要两次地址转换才能访问到宿主机中的真正物理内存，而宿主机内存寻址只需要一次地址转换. 需要确保不同系统中的文件虚拟地址空间在不同的地址转换机制下能够正确地访问到文件数据物理页.

（2）虚拟机用户和宿主机用户都可以通过文件系统提供的接口去访问文件系统中的资源，需要确保多个不同系统的进程并发地访问共享内存文件系统 StargateFS 中的临界资源.

（3）虚拟机和宿主机 VFS 层都提供了缓存机制. 当更新操作提交到 VFS 层后，VFS 层并没有立即把缓存信息更新到文件系统中. 需要确保多个系统 VFS 缓存中的信息与文件系统中的信息一致.

下面介绍解决这些问题相关关键技术，以及共享内存文件系统 StargateFS 的实现.

3.2　内存共享

在虚拟机同驻的环境下，共享内存可以提升虚拟机间、虚拟机与宿主机间的数据传输的效率. 为了实现面向同驻虚拟机间高效的共享内存文件系统，StargateFS 使用共享的模拟非易失性内存作为存储空间.

在 KVM 平台中，每台虚拟机对应宿主机中一个 QEMU 用户进程[23]，虚拟机的物理地址空间是 QEMU 用户进程虚拟地址空间中声明的一段虚拟的地址空间. 当虚拟机真正存放数据时，KVM 通过缺页中断把 QEMU 进程该段虚拟地址空间（虚拟机物理地址空间）映射到宿主机真正的物理页. 因此，可以在所有虚拟机的物理地址空间中都预留一段相同大小的内存空间，使得该段内存空间都映射到宿主机上一段相同大小、连续的物理地址空间，以实现虚拟机间的内存共享. 另一方面，宿主机有权限访问虚拟机共享内存所对应的宿主机物理页，在宿主机中把这些物理页映射到宿主机进程的虚拟地址空间，能够实现虚拟机与宿主机之间的内存共享.

如图 4 所示，在虚拟机 1 和虚拟机 2 中分别预留了一段固定的物理地址空间，分别映射到宿主机中同一段预留的真实物理内存空间，实现虚拟机间的内存共享. 一般情况下，虚拟机物理地址到宿主机物理地址间的映射关系是随机的. 当虚拟机进程向虚拟机物理内存写入数据时，如果不存在虚拟机物理地址到宿主机物理地址第二次地址转换映射关系，CPU 就会产生缺页错误，该缺页错误由宿主机

中 KVM 模块处理，宿主机中 KVM 模块会随机申请一段空闲的物理地址空间，建立虚拟机物理地址到宿主机物理地址的转换关系. 而在 StargateFS 中，宿主机中预留的物理地址空间在宿主机启动时已分配完成. 因此，直接把虚拟机中预留的物理地址空间线性映射到宿主机中预留的物理地址空间. 例如，虚拟机 1 和虚拟机 2 分别从 4 GB 开始，预留 8GB大小的物理内存空间，线性映射到宿主机从 16GB开始预留的 8GB 大小的物理内存空间，虚拟机中预留物理空间的开始物理地址 0x100000000 线性映射到宿主机预留的物理空间起始物理地址 0x400000000.

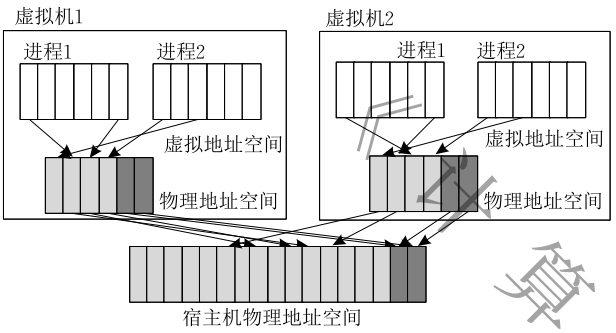


图 4 共享内存实现

同时，在宿主机中把虚拟机共享内存所对应的宿主机物理页映射到宿主机进程的虚拟地址空间，实现虚拟机宿主机间的内存共享. 在虚拟机和宿主机系统中分别增加一个新的大小相同的内存管理区 ZONE_SM 作为共享内存，由 StargateFS 管理共享内存的分配和回收. ZONE_SM 的大小是用户可配置的. 宿主机上往往运行有一个或多个虚拟机，应先为宿主机系统预留足够的物理内存空间，再根据用户的需求设置 ZONE_SM 的大小.

3.3 文件共享

在介绍文件共享之前，首先介绍共享内存文件系统 StargateFS 中文件的索引结构共享文件页表，然后介绍在虚拟机同驻的环境下，如何实现虚拟机间、虚拟机和宿主机间的文件共享.

为有效地利用 MMU 加速共享文件数据页的索引，StargateFS 使用可共享的页表结构作为文件数据的索引结构. 该索引结构称为“文件页表[18]”. 共享文件页表的结构与 Linux 系统的页表相似. 64 位 Linux 系统通常采用 4 级分页模型实现内存寻址，所以共享文件页表的结构可以是 1 到 4 级，其具体级数可根据需要支持的文件最大大小进行配置. 不同级数共享文件页表可支持的文件大小计算方法如下：

$$MaxFileSize=\left(\frac{PageSize}{PageEntrySize}\right)^{N}\cdot PageSize,$$

其中 $MaxFileSize$ 表示最大文件大小，$PageSize$ 表示页大小，$PageEntrySize$ 表示页表项的大小，N 表示共享文件页表的级数. 在当前的设计中，采用 3 级共享文件页表的索引结构组织共享文件数据物理页，页大小为 4KB，页表项大小为 8Bytes，如图 5 所示. 3 级共享文件页表最大支持 512 GB 大小的文件. 在当前的实现中限制每个文件的最大大小为 32GB. 在文件创建时，建立文件的共享文件页表，并在 StargateFS 文件索引节点中持久化保存共享文件页表 L3 级页表页的基地址. 打开文件时，把使用 L3 级页表页的基地址构成的页表项插入到内核页表的 PGD 级中，使得文件数据可以映射到虚拟地址空间中.

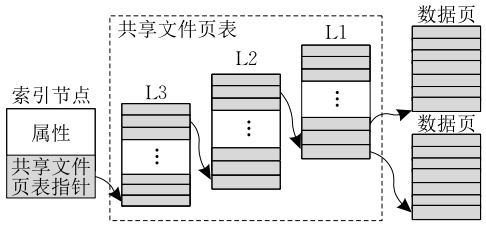


图 5 共享文件页表

为了支持虚拟机和宿主机中的进程访问文件，在宿主机中打开文件时，把共享文件页表插入宿主机中进程的页表中；在虚拟机中打开文件时，把共享文件页表插入虚拟机中进程的页表中.

然而，在当前主流的 x86 架构中，硬件辅助内存虚拟化提供的 EPT(Extended Page Table)[24] 机制直接在硬件层面上支持虚拟机虚拟地址 GVA 到虚拟机物理地址 GPA，再到宿主机物理地址 HPA 的两次地址转换. 因此，在宿主机上用于宿主机虚拟地址 HVA(Host Virtual Address)到宿主机物理地址 HPA 一次地址转换的共享文件页表不能直接插入到虚拟机中的进程页表. 宿主机中的进程页表页表项中的页框地址代表宿主机页框号 HFN(Host Frame Number)，而虚拟机中的进程页表页表项中的页框地址代表虚拟机页框号 GFN(Guest Frame Number)，即宿主机 QEMU 进程虚拟地址空间的一个页号. 当把共享文件页表插入到虚拟机中的进程页表时，虚拟机内存寻址会把宿主机物理地址当作是虚拟机物理地址，将导致虚拟机内存寻址错误.

为了在虚拟机中的两次地址转换和在宿主机中的一次地址转换都能利用虚拟地址空间正确地访问

共享文件数据，考虑两种思路解决该问题：一种是在共享内存文件系统中分别为虚拟机和宿主机维护不同的文件页表，确保虚拟机文件页表和宿主机文件页表都能正确地访问文件数据；另一种思路是直接把共享文件页表插入到虚拟机中的进程页表中，修改 EPT 页表项使得共享内存中虚拟机物理地址和宿主机物理地址一一对应，且属于这段内存空间的虚拟机物理地址经过 EPT 转换后的值不变. 第一种思路构建虚拟机文件页表需要访问宿主机的文件页表，且同一个文件两种文件页表一致性维护开销大. 因此，本文采用第二种方式修改 EPT 的页表项，实现虚拟机宿主机间的文件共享.

为使 GFN 经 EPT 转换后的值不变，需要设计新的共享内存管理方法. 主要有两个方面：(1) 确保虚拟机共享内存的起止物理地址与宿主机共享内存的起止物理地址相同；(2) 重新设计 KVM 模块中 EPT 页表 PTE(Page Table Entry)级页表项的构建函数，使得虚拟机共享内存中的物理页与宿主机共享内存中的物理页一一对应，如图 6 所示. 例如，虚拟机需要往共享内存中 GFN 为 0x1000000 的虚拟机物理页写入数据，经过 EPT 页表转换后，实际写入数据的 HFN 仍为 0x1000000. 以上两个设计使得无论是在虚拟机中文件数据索引两次地址转换，还是在宿主机中一次地址转换，StargateFS 都能利用硬件 MMU 加速定位文件数据物理页.

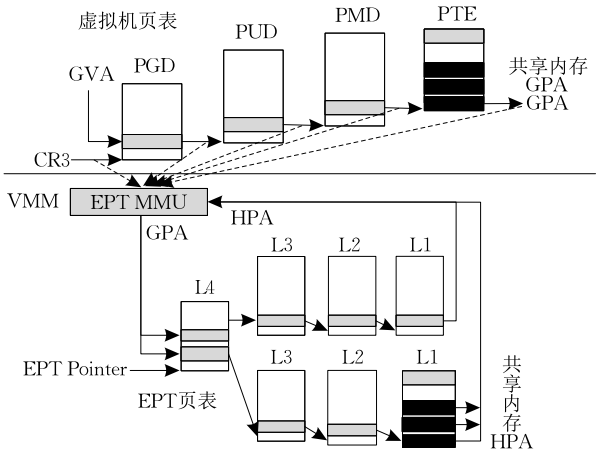


图 6　EPT 页表项修改

3.4　预留虚拟地址空间管理

在 Linux 中，进程的虚拟地址空间被划分为两个部分，一部分是进程无论运行在用户态还是内核态都可以寻址，另一部分是进程只有运行在内核态才能寻址，即内核虚拟地址空间. StargateFS 分别在虚拟机和宿主机内核虚拟地址空间段划分一段系统预留的虚拟地址空间用于访问共享文件. 如图 7 所示，预留虚拟地址空间被分成两段，一段是线性映射区，另一段是动态映射区. 挂载 StargateFS 时，把超级块、索引节点区、共享信号量区所在的物理地址空间线性映射到虚拟机和宿主机中的虚拟地址空间线性映射区. 虚拟地址空间动态映射区在打开文件时为文件分配虚拟地址空间.

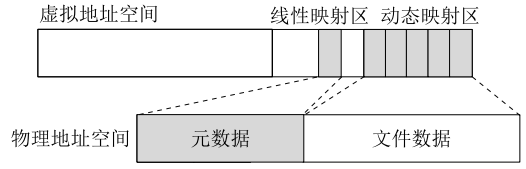


图 7　预留虚拟地址空间

动态映射区被划分为多个相同大小的连续虚拟地址空间块，称为“文件虚拟地址空间[18]”. 文件虚拟地址空间的大小等同于文件系统支持的最大文件尺寸. 动态映射区使用动态分配回收策略管理虚拟地址块. 当在宿主机中打开共享文件时，从宿主机中的动态映射区分配一段虚拟地址空间，用文件的索引结构将共享文件数据映射到该虚拟地址空间. 同样地，当在虚拟机中打开共享文件时，从虚拟机中的动态映射区分配一段连续的虚拟地址空间. 当关闭文件时，回收该文件使用的虚拟地址空间. 文件只在被打开时占有虚拟地址空间. 使用动态分配回收策略管理文件的虚拟地址空间可以避免固定分配虚拟地址空间的浪费；每次分配文件系统支持最大的虚拟地址空间，能够避免文件大小增长时文件的虚拟地址空间增长的冲突.

3.5　元数据同步

StargateFS 的数据存放在共享内存中，无论是虚拟机进程还是宿主机进程都可以使用文件系统提供的接口访问其数据. 访问文件系统的数据一般遵循两种操作流程，即“只读”及“读修改写”. 当多个进程顺序访问共享内存文件系统时，不会导致共享资源竞争访问的问题. 而一台虚拟机多个进程或多台虚拟机的多个进程并发访问文件系统时，情况变得复杂. 对于单台虚拟机上多个进程并发访问文件系统的情况，Linux 内核提供了许多同步机制如原子操作、自旋锁、信号量等. 对于属于不同虚拟机的多个进程和属于宿主机的进程(属于不同系统的多个进程)并发访问共享内存文件系统的情况，不能直接使用 Linux 内核提供的用于单系统的同步机制.

属于不同系统的多个进程并发访问共享内存文件系统时，需要保护的共享资源有两类：元数据和文

件数据.文件数据的同步将在第 4 节讨论.元数据包括超级块信息、索引节点信息、目录项信息、空闲空间链表等.不同系统的多个进程并发访问共享元数据时可能导致两方面的问题.一方面是共享元数据竞争访问的同步问题.需要考虑同步的元数据包括两类:一类是文件系统元数据,包括超级块信息、空闲数据块链表、空闲索引节点;另一类是文件元数据,包括文件及目录文件的索引节点信息、共享文件页表结构.另一方面,无论是虚拟机系统还是宿主机系统,VFS 层都提供了缓存机制,当虚拟机上的进程更新文件元数据信息后,没有及时同步到其他虚拟机或宿主机 VFS 的缓存中,导致不同系统上的元数据信息不一致.

对于竞争访问共享元数据的问题,Linux 内核提供关中断、每 CPU 变量、原子操作、自旋锁、信号量、互斥量等机制,这些机制在单台系统中可以有效地解决所有同步问题[25].然而,内核同步机制只能保证一个系统中多个内核控制路径对共享元数据的安全访问,不能保证多个系统间对共享元数据的安全访问.虚拟机同驻在一台宿主机上时,把锁变量存放在共享内存中的共享锁机制能够保证不同系统内核控制路径对共享元数据互斥访问.

同步机制的实现包含两种方式:轮询(Polling)和阻塞(Blocking).轮询如自旋锁机制,进程循环地测试资源是否可用,循环测试可能会降低 CPU 的效率.阻塞如互斥量机制,阻塞进程执行直到资源可用.为了避免 CPU 资源的浪费,StargateFS 优先尝试使用共享互斥锁控制不同系统内核控制路径对共享元数据互斥访问.当不同系统中的进程同时访问共享资源时,共享互斥锁使得同一时刻只有一个系统的内核控制路径进入临界区执行,阻塞其它请求共享临界区的虚拟机或宿主机用户进程,并把阻塞进程添加到自己的等待队列中.当共享互斥锁被释放时,根据特定的算法唤醒等待队列中的阻塞进程进入临界区执行.然而,在共享互斥锁唤醒等待队列中的阻塞进程时可能会导致地址访问错误,其原因在于共享互斥锁等待队列中的进程可能属于不同虚拟机或宿主机的进程.

定义在 StargateFS 共享内存中的共享自旋锁可以控制不同系统多个进程互斥地访问共享元数据资源且不会导致错误.不同系统的进程访问共享自旋锁时会访问到同一物理地址空间.在 x86 架构下通过锁前缀对总线加锁的方式,保证多个系统内核控制路径原子地修改共享自旋锁锁变量的值.

对于元数据一致性问题,虚拟机和宿主机系统的 VFS 层提供了索引节点高速缓存机制、目录项高速缓存机制.目录项高速缓存用于提高目录项对象的处理效率,其缓存内容不写回到文件系统.因此,需要考虑索引节点信息的一致性.当虚拟机 1 中的进程访问文件更新了虚拟机 1 中 VFS 缓存的文件索引节点信息,由于 VFS 层的缓存回写机制并没有立即把更新后的索引节点信息写入 StargateFS 中,导致虚拟机 1 上 VFS 中文件索引节点信息与虚拟机 2 上 VFS 中的文件索引信息不一致.当虚拟机 2 中的进程访问文件时,访问到的文件索引节点信息可能是旧的信息.

解决这个问题的方法有两种:一种是像对待文件数据一样,使用 O_DIRECT 模式,绕过 VFS 层提供的缓存机制,对文件索引节点信息的修改直接更新到共享文件系统中;另一种是在文件读写过程中主动地把 VFS 缓存中的索引节点信息同步到文件系统中,确保每次访问索引节点信息都是最新的.第一种方式需要对虚拟机操作系统的 I/O 路径做大量的修改,难以应用到诸多现有虚拟机的操作系统中.

因此,StargateFS 采用第二种方法来控制不同系统 VFS 层元数据的一致性.在文件打开、读写时,StargateFS 主动把文件索引节点信息同步到 VFS 层缓存索引节点中.在文件读写后,关闭文件时,StargateFS 主动把 VFS 层缓存索引节点信息更新到文件索引节点中,保证在文件读写过程中,虚拟机、宿主机 VFS 中缓存信息和文件系统中的元数据信息是一致的.

3.6 文件系统实现

采用提出的新型共享内存文件系统设计,在 Linux 系统中基于 KVM 实现高效共享内存文件系统 StargateFS.为了支持 StargateFS,对虚拟机、宿主机中 Linux 系统进行以下 3 个方面的修改:

(1)通过 DRAM 模拟非易失性内存为共享内存文件系统提供存储空间.修改宿主机、虚拟机 Linux 系统内核中的内存管理模块,分别从物理地址空间 16 GB 开始增加一个新的大小为 8 GB 的内存管理区 ZONE_SM;

(2)为实现虚拟机间、虚拟机宿主机间的共享内存,修改宿主机 Linux 内核中 KVM 模块,把虚拟机中 ZONE_SM 物理地址空间线性映射到宿主机中 ZONE_SM 物理地址空间;

(3)在宿主机、虚拟机中为 StargateFS 分别预留一段内核未使用的虚拟地址空间,用于访问共享

文件数据.

StargateFS 采用 C 语言编写，在 Linux 内核中实现. 按照 Linux 虚拟文件系统 VFS 通用文件模型标准分别实现文件系统、文件以及目录的常规操作，如文件系统的安装、卸载等操作，文件的打开、读写、关闭、删除等操作，文件属性的查询、修改等操作. StargateFS 包含文件数据管理子系统，元数据管理子系统，存储空间管理子系统. 存储空间管理子系统实现 StargateFS 物理空间的数据布局和管理，负责空闲物理页、文件虚拟地址空间的分配和回收. 当用户安装 StargateFS 时，在 *fill_super* 函数中初始化物理空间的数据布局，并分别填充 StargateFS 中和 VFS 中的 *superblock* 结构体. 文件数据管理子系统向 VFS 层提供操作文件的接口，并实现文件的基本操作. 文件元数据管理系统向 VFS 层提供操作文件系统元数据、文件元数据的接口，并实现文件系统元数据、文件元数据的操作.

StargateFS 采用存放在共享内存中的共享自旋锁控制不同系统多个进程互斥地访问共享元数据资源. 在共享元数据结构体中增加共享自旋锁字段，当多个虚拟机内核控制路径同时访问同一共享元数据资源时，每个虚拟机内核控制路径获取一个排队号，并循环地比较自己的排队号与当前共享元数据资源结构体中共享自旋锁的号是否相等，如相等，则访问共享元数据资源.

StargateFS 是在 Linux 内核中实现的，虚拟机用户、宿主机用户只能通过文件系统已实现的文件操作系统调用访问 StargateFS 中的资源. StargateFS 使用虚拟机间、虚拟机宿主机间的模拟共享非易失性内存作为存储空间，由 StargateFS 管理空间的分配和回收，不会影响宿主机系统或虚拟机系统内存空间的管理. 用于访问共享文件的虚拟地址空间是内核预留的虚拟地址空间，文件数据的访问直接由硬件执行，StargateFS 不会影响虚拟机系统或宿主机系统中数据的安全性. 当 StargateFS 安装完成后，虚拟机用户、宿主机用户都有读、写、执行权限操作共享文件，在以后的工作中，我们将实现共享文件对不同系统的用户开放不同的访问权限.

3.7　性能分析

在虚拟机同驻的环境中，虚拟机用户共享访问宿主机上或其他虚拟机上的文件时，访问共享文件的性能开销主要源于两个方面：(1) 虚拟机 VFS 层、宿主机 VFS 层以及共享方法各个软件层次的软件流程执行开销；(2) 共享文件数据多次内存拷贝的开销.

根据虚拟机用户访问共享文件的性能开销的来源，分析 StargateFS 文件读写性能的优势. 以 VirtFS 作为对比对象，用 L_{VirtFS} 和 $L_{\mathrm{StargateFS}}$ 分别表示虚拟机用户访问 VirtFS 共享宿主机 EXT4-DAX 中共享文件的时间开销和 StargateFS 中共享文件的时间开销，L_{VirtFS} 和 $L_{\mathrm{StargateFS}}$ 可表示为

$$L_{\mathrm{VirtFS}}=L_{\mathrm{GV}}+L_{\mathrm{VirtFSC}}+L_{\mathrm{VirtIO}}+L_{\mathrm{VirtFSS}}+L_{\mathrm{HV}}+3\cdot L_{\mathrm{M}},$$

$$L_{\mathrm{StargateFS}}=L_{\mathrm{GV}}+L_{(\mathrm{GV,GP})}+L_{(\mathrm{GP,HP})}+L_{\mathrm{M}}.$$

其中，L_{GV} 表示访问共享文件时虚拟机中 VFS 层的时间开销，L_{VirtFSC}、L_{VirtIO}、L_{VirtFSS} 分别表示 VirtFS 共享文件方法中 VirtFS 客户端、VirtIO 以及 VirtFS 服务端的时间开销，L_{HV} 表示访问共享文件时宿主机中 VFS 层的时间开销. $L_{(\mathrm{GV,GP})}$ 表示虚拟机虚拟地址转换为虚拟机物理地址的时间开销，$L_{(\mathrm{GP,HP})}$ 表示虚拟机物理地址转换为宿主机物理地址的时间开销. L_{M} 表示虚拟机中内存拷贝的时间开销. 对比 L_{VirtFS} 和 $L_{\mathrm{StargateFS}}$，访问两种共享文件系统中共享文件的时间开销主要由 $L_{(\mathrm{GV,GP})}$、$L_{(\mathrm{GP,HP})}$、L_{M} 与 L_{VirtFSC}、L_{VirtIO}、L_{VirtFSS}、$3L_{\mathrm{M}}$ 决定. 而 $L_{(\mathrm{GV,GP})}$、$L_{(\mathrm{GP,HP})}$ 由硬件 MMU 决定，L_{VirtFSC}、L_{VirtIO}、L_{VirtFSS} 由 VirtFS 共享方法中的各个软件层次决定，各个软件层次间存在多次内存数据拷贝. 硬件 MMU 执行地址转换的时间在纳秒级别，软件流程执行时间在微秒级别，硬件直接执行的时间开销远小于软件流程执行的时间开销. 因此，StargateFS 文件访问速度理论上比 VirtFS 快.

对比 VirtFS，StargateFS 的性能优势来源于：

(1) StargateFS 直接使用虚拟机间、虚拟机宿主机间共享的模拟非易失性内存作为存储空间，在虚拟机看来，StargateFS 就是本地内存文件系统，减少了共享文件访问的 I/O 软件层次和共享文件数据的内存拷贝次数；

(2) StargateFS 使用共享文件页表组织共享文件数据页，利用文件虚拟地址空间和硬件 MMU 加速索引共享文件数据页，避免了软件查找共享文件索引的时间开销问题.

3.8　文件系统和文件操作

3.8.1　文件系统的安装和卸载

当虚拟机宿主机间的共享内存建立完成后，使用 *mount* 命令在虚拟机和宿主机中安装 StargateFS，虚拟机用户和宿主机用户都可以像访问本地文件系统一样访问 StargateFS. 为避免新加入的虚拟机在安装文件系统时格式化共享内存中的数据，在超级块中增加了安装计数器. 当计数器的值大于等于 1 时，新加入的虚拟机在安装文件系统时不对共享内

存进行格式化.文件系统根据共享内存起始物理地址和安装点信息,在内核中创建并初始化安装点和超级块对象,并完成文件系统超级块对象、索引节点区、空闲数据页、文件系统根目录等结构的初始化.在卸载文件系统时,使安装计数器减 1.

3.8.2 文件操作

具体步骤如下:

(1)打开操作.当用户进程调用 *open* 系统调用打开文件时,为文件分配一段文件虚拟地址空间.如文件已创建,将共享文件页表插入到虚拟机或宿主机的内核页表中,返回文件描述符;如果被打开的文件未创建,首先通过文件创建接口分配一个新的索引节点,新申请物理页构建文件的共享文件页表,然后将共享文件页表插入到内核页表中,返回文件描述符;

(2)读操作.当用户进程调用 *read* 系统调用读文件数据时,通过传入的文件描述符和打开模式参数,获取文件虚拟地址空间的起始地址和当前文件偏移量.通过文件虚拟地址空间和共享文件页表利用 MMU 加速定位文件数据物理页,并将文件数据拷贝到用户缓冲区.文件数据没有经过传统的页缓存.定位文件数据页时不需要搜索文件数据索引结构,仅需要计算起始虚拟地址即可;

(3)写操作.当用户进程调用 *write* 系统调用写入数据时,根据文件当前大小与当前文件偏移量的差值和写入数据大小的关系,把写操作分为 3 种类别:就地更新、追加写、部分追加写.当写入起止位置被包含在文件数据大小内时,即就地更新,直接将用户缓冲区的数据拷贝到文件数据页中.当写入方式为追加写和部分追加写时,首先申请新的文件数据页,然后更新共享文件页表,最后将用户缓冲区的数据拷贝到文件数据页中;

(4)关闭操作.当用户进程调用 *close* 系统调用关闭文件时,首先检查文件引用计数器是否为 0.如果引用计数器为 0,表明没有用户进程在访问该文件,从内核页表中删除共享文件页表对应的 PGD 级页表项,回收该文件的虚拟地址空间;如果文件引用计数器的值不为 0,引用计数器减 1.

4 共享内存信号量

在 3.5 节中讨论了不同系统内核控制路径对文件系统共享元数据的同步访问,本节讨论提供一种同步机制,使得不同系统多个用户进程间同步地访问同一个共享文件的文件数据.StargateFS 中的文件默认以 O_DIRECT 模式打开,读写文件数据时不经过 VFS 层的数据块缓存,使得共享文件数据的同步变得简单.

在一个系统中,两个用户进程以读写模式同时打开相同文件并写入数据时,第一个进程写入的数据将会被第二个进程写入的数据覆盖,这是由于每个进程对应该文件都有一个文件对象,其中保存了当前文件偏移量.同样,属于多个虚拟机系统、宿主机系统的多个进程同时打开 StargateFS 中相同文件并写入数据时,先写入的数据也会被后写入的数据覆盖.另外,在一些应用场景如生产者消费者问题、读者写者问题中,多个进程需要同步地读写同一个共享文件的数据.

在以上应用场景中,需要提供一种机制,使得多个进程同步地读写共享文件.在单系统中,用户进程间同步的实现方式分为 2 种:一种是完全依赖内核实现,如 System V IPC[26] 信号量机制,向用户空间提供系统调用接口访问信号量.另一种是在用户空间实现大部分同步操作,仅少部分操作经过代价较大的系统调用在内核态完成,如 Futex[27].当属于不同系统的多个用户进程同时读写相同文件时,需要提供一种新的机制,使得不同系统上的进程受控读写同一个文件.

实现多系统进程间对同一共享文件同步访问有两种方法:一种是通过共享内存实现同步机制,另一种是通过消息机制[28].由于所有虚拟机都位于同一台宿主机上,通过共享内存实现不同虚拟机、宿主机用户进程间的同步.实现共享内存同步机制的一种简单途径是创建一个共享文件并对共享文件进行加锁解锁.这种方式通过进程占有共享文件实现同步的代价非常高.另一种是提供系统调用直接访问共享内存,不涉及文件系统的访问.本文采用第二种方式实现基于共享内存的多系统进程间的同步机制,分别在虚拟机和宿主机系统中增加一套访问共享内存的系统调用.

为了在多虚拟机、宿主机进程间提供一种有效的基于共享内存的同步机制,StargateFS 在虚拟机宿主机共享的内存中预留了一段物理内存空间存储共享信号量,如图 8 所示.共享信号量区的物理地址空间分别映射到虚拟机和宿主机中预留虚拟地址空间线性映射区.通过共享信号量区的起始虚拟地址和共享信号量的编号可以快速地定位每个共享信号量空间.空闲共享信号量空间使用链表管理,利用

3.5 节中所述的共享自旋锁来控制不同系统内核线程对共享链表的互斥访问.为了满足不同应用场景需求,在用户空间和内核空间分别实现共享内存信号量变量的操作.

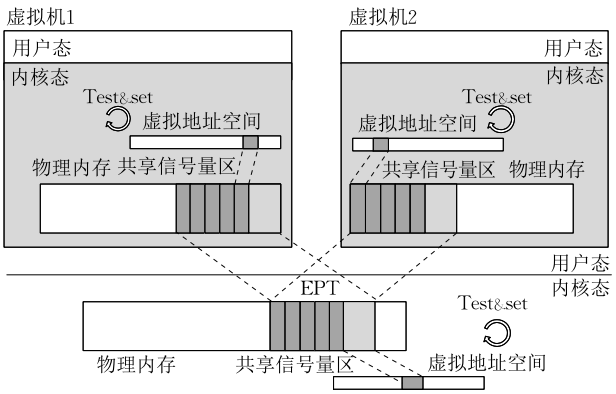


图 8　共享信号量实现

通过把共享信号量变量存放在不同虚拟机、宿主机用户进程间共享的内存中实现用户空间共享内存信号量,支持轮询同步机制.例如,直接把 Pthread[29] 库提供的自旋锁结构 Pthread_spinlock_t 定义在共享内存中实现多个不同系统的用户进程间同步访问同一共享文件的数据.因此,用户空间共享内存信号量的实现需要在虚拟机和宿主机系统中分别新增建立共享内存的系统调用,以允许不同系统用户进程从共享信号量区分配共享信号量存储空间,并映射到用户空间.

内核空间信号量直接在内核中修改共享内存信号量变量的值,并向用户进程提供信号量申请、操作及释放系统调用接口.当不同系统的多个进程通过内核空间共享内存信号量互斥访问同一共享文件资源时,先检测共享信号量的值,判断共享文件资源是否可用,可用则修改共享信号量计数器的值.共享信号量计数器值的检测修改是"读修改写"操作,需保证该组操作的原子性,以避免多个进程同时检测到共享文件资源可用.一种方式是使用 3.5 节所介绍的共享自旋锁,控制同一时刻仅有一个内核控制路径在检测并修改共享信号量的值.另一种方式是使用 Test&set[30] 设置并返回原值原子操作指令控制,如图 8 中所示.共享文件资源可用时共享信号量计数器的值为 0,共享文件不可用时计数器的值为 1.仅当成功把计数器值设为 1 且返回计数器原值为 0 的进程允许访问共享文件资源.由于 Test&set 是原子操作指令,多个进程竞争时,仅有一个进程的返回值为 0,其它进程的返回值为 1,返回值为 1 的进程循环测试.因此,Test&set 指令可避免多个进程同时检测到共享文件资源可用错误修改共享信号量值的问题.

5　性能测试

本章测试 StargateFS 的性能,并分析测试结果.分别从文件读写吞吐量和文件共享延迟两个方面评估 StargateFS 的性能.使用标准测试工具 FIO 测试不同共享文件系统的吞吐量;通过在不同虚拟机上运行自定义程序,先写后读同一共享文件的时间开销来测试共享延迟.另外,还测试了共享信号量申请释放的时间开销.

5.1　实验环境配置

我们以 DELL T7910 16 核工作站作为宿主机完成测试实验.宿主机硬件环境为 Intel(R) Xeon(R) E5-2630 v3@2.40 GHz 处理器,容量 128 GB 频率 2133 MHz 内存,Intel i210 Gigabit 网卡.宿主机系统为 Ubuntu 14.04,内核版本为 Linux 3.11.8,KVM 为 2.0.0 版本,QEMU 为 2.0.0 版本.在未特别说明的情况下,每个虚拟机分配 8 个虚拟处理器 vCPU,32 GB 内存,E1000 模拟网卡.虚拟机系统为 Ubuntu 14.04,内核版本为 Linux 3.11.8,虚拟机通过 NAT 网络地址转换模式接入网络.

在宿主机上,从物理内存 16 GB 开始预留 8 GB 大小的空间模拟新型非易失性内存.在宿主机和虚拟机系统中,分别从 16 GB 物理内存开始,新增 8 GB 大小的内存管理区 ZONE_SM.修改宿主机 KVM 模块把虚拟机 ZONE_SM 中的物理地址空间线性映射到宿主机 ZONE_SM 中的物理地址空间,完成虚拟机间、虚拟机与宿主机间共享内存的配置.

5.2　读写性能测试

实验使用 FIO 工具对比测试 StargateFS 与目前基于 KVM 性能最优的共享文件系统 VirtFS[14]、传统的共享文件系统 Samba、NFS[12] 分别共享内存文件系统 EXT4-DAX 的文件读写吞吐量.

EXT4-DAX 是 Linux 为了支持非易失性内存而扩展 EXT4 实现的内存文件系统,它绕过页缓存直接读写非易失性内存设备.经测试,EXT4-DAX 单线程文件读写最大吞吐量为 6649.4 MB/s,虚拟机间、虚拟机与宿主机间通过虚拟网卡和 VirtIO 通信的最大吞吐量分别为 250.3 MB/s、2585.6 MB/s.因此,使用宿主机上的内存文件系统 EXT4-DAX 进行测试不会影响共享方法的文件读写性能测试.

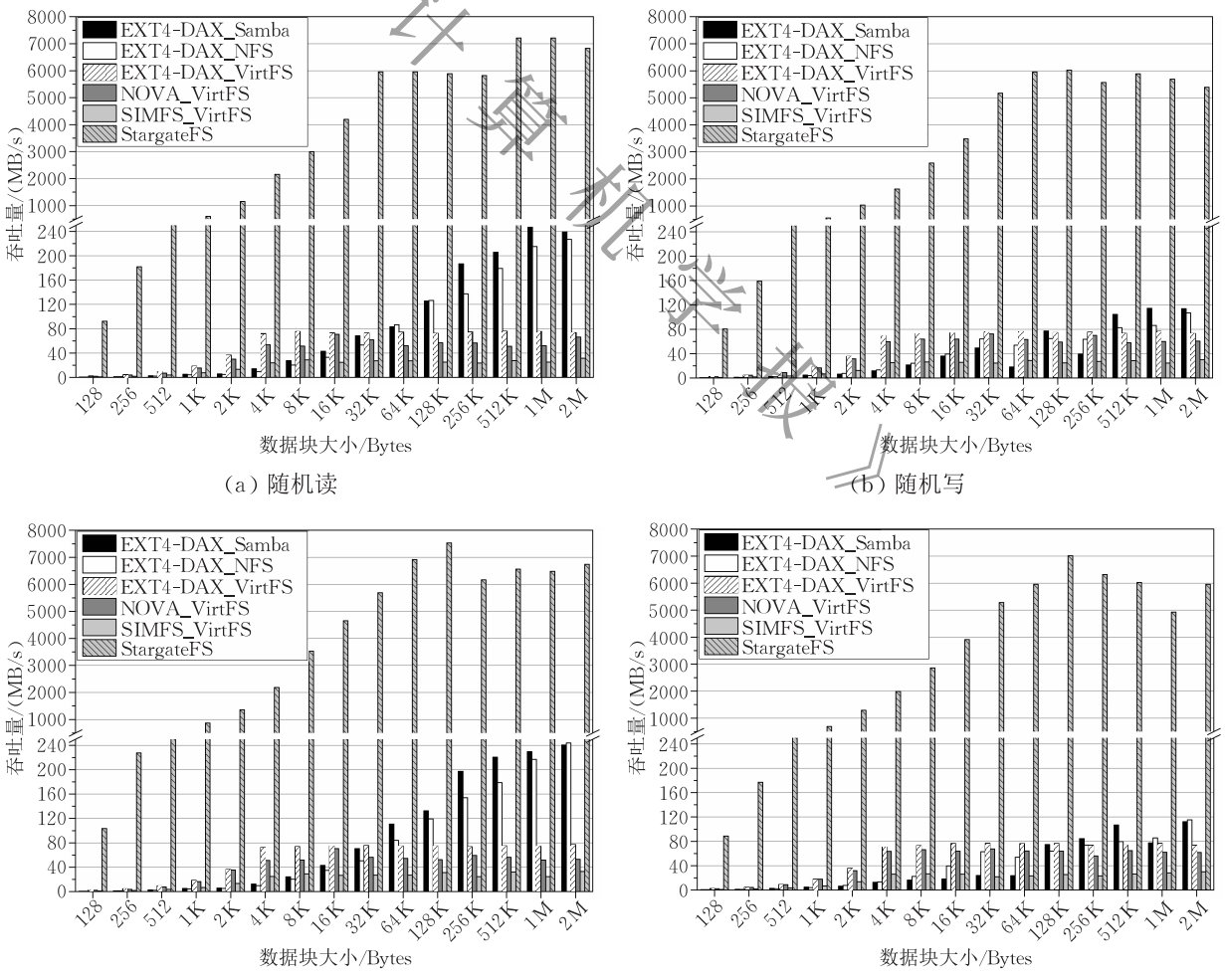
实验中使用预留的内存管理区 ZONE_SM 作为 StargateFS 的存储空间，在宿主机和虚拟机中使用 *mount* 命令安装 StargateFS 文件系统. 在宿主机中配置 8 GB 大小的 RAMDisk① 作为 EXT4-DAX 的存储空间. RAMDisk 把内存模拟为内存设备. 使用 EXT4 文件系统类型格式化 RAMDisk，通过文件系统挂载选项-o dax 将已格式化为 EXT4 的 RAMDisk 挂载到目录/mnt/share.

Samba、NFS 使用模拟网卡 E1000 与宿主机通信，VirtFS 使用 Virtio 与宿主机通信. Samba 的版本为 4.3.11，在宿主机上配置 Samba 服务端共享宿主机目录/mnt/share，客户机对共享目录可读写，其他参数使用默认配置. 重启 Samba 服务，即可在虚拟机上通过 IP 和共享目录名安装宿主机上共享的目录到虚拟机. NFS 采用 NFSv4 版本，配置与 Samba 同样的文件访问权限. VirtFS 使用 Squash 模式共享宿主机目录/mnt/share，传输消息最大报文大小采用默认配置 8 KB.

在以上实验配置完成后，在虚拟机中使用 FIO 工具分别进行单线程、多线程文件读写性能测试. 下面先对比单线程文件读写性能，然后分析多线程文件读写性能.

5.2.1 单线程文件读写性能

在虚拟机中使用 FIO 工具测试单线程随机读写、顺序读写 Samba、NFS、VirtFS 以及 StargateFS 中文件的性能，测试文件大小为 512 MB. 单线程文件读写性能测试结果如图 9 所示，图中数据块大小表示 FIO 每次请求的 I/O 块大小. 实验结果表明，StargateFS 单线程文件读写性能在所有测试用例中都优于 Samba、NFS、VirtFS 共享方法，平均吞吐量比 Samba、NFS、VirtFS 共享宿主机 EXT4-DAX 文件系统分别快 117.7 倍、121.6 倍、59.3 倍.



(a) 随机读 (b) 随机写

(c) 顺序读 (d) 顺序写

图 9 单进程文件读写性能

StargateFS 单线程文件顺序读吞吐量分别比 Samba、NFS、VirtFS 共享 EXT4-DAX 快 28.0 至

① RAMDisk [Online]. https://www.kernel.org/doc/Documentation/blockdev/ramdisk.txt, 2004.10.22

234.0 倍、27.5 至 273.9 倍、30.0 至 100.8 倍，顺序写吞吐量分别快 53.0 至 253.1 倍、51.5 至 185.1 倍、28.1 至 91.7 倍；单线程随机读吞吐量分别比 Samba、NFS、VirtFS 共享 EXT4-DAX 快 28.5 至 199.6 倍、30.0 至 239.6 倍、29.9 至 95.6 倍，随机写吞吐量分别快 47.3 至 329.1 倍、50.4 至 169.1 倍、23.4 至 81.1 倍.

单线程文件读写性能测试结果验证了 3.7 节中的分析结果. 实验测试结果表明，StargateFS 共享文件的性能优于 Samba、NFS、VirtFS 共享方法. StargateFS 允许虚拟机用户直接访问共享内存中的文件，避免了诸多软件层次和文件数据拷入拷出缓冲区的开销，并利用硬件 MMU 加速索引文件数据页，共享文件的读写性能得到很大的提升.

目前，国际上存在很多先进的新型非易失性内存文件系统，如 NOVA[19]、SIMFS[18]. 这些文件系统是本地内存文件系统，而 StargateFS 是基于新型非易失性内存面向同驻虚拟机的共享内存文件系统，其最大的特征在于虚拟机间、虚拟机宿主机间高效地共享文件. 因此，我们使用 FIO 工具对比测试 StargateFS 与 VirtFS 分别共享宿主机上内存文件系统 NOVA 和 SIMFS 的单线程文件读写性能. 在宿主机上使用 DRAM 分别模拟 8GB 大小的非易失性内存设备作为 NOVA、SIMFS 的存储空间，使用 VirtFS 共享宿主机上的内存文件系统 NOVA、SIMFS.

实验测试结果如图 9 所示，StargateFS 单线程平均吞吐量分别比 VirtFS 共享 NOVA、SIMFS 快 73.5 倍、164.6 倍. 实验结果同样说明使用共享方法 VirtFS 在虚拟机间、虚拟机宿主机间共享文件的性能开销大，共享方法的读写带宽限制了共享文件的读写性能. 而 StargateFS 使用虚拟机间、虚拟机宿主机间的共享内存作为存储空间，并利用硬件 MMU 加速定位共享文件数据页，减少共享文件访问的软件层次和软件流程查找共享文件索引的开销，有效地提升共享文件数据的访问效率.

本文分析了 StargateFS 中的文件数据映射和内存映射 MMAP[31] 的区别. MMAP 把对用户态虚拟地址空间中某个字节的访问转换成对文件中相应字节的操作. 虽然 MMAP 和 StargateFS 中的文件数据映射都是将打开文件的文件数据物理页映射到虚拟地址空间访问，但是两者有本质的区别：(1) StargateFS 是兼容 POSIX 的共享内存文件系统，它将文件数据页映射到预留的内核虚拟地址空间，同时支持把共享文件数据物理页映射到宿主机内核虚拟地址空间和虚拟机内核虚拟地址空间. 而 MMAP 是一个系统调用接口，只支持在一个系统中映射文件数据页到用户虚拟地址空间；(2) StargateFS 支持 *mmap* 方法，当使用 *mmap* 方法将 StargateFS 中打开的文件映射到用户态虚拟地址空间后，对文件数据的访问是内存访问. 此外，我们尝试在虚拟机中使用 MMAP 映射 VirtFS 共享的宿主机上的 EXT4-DAX 文件，返回操作不允许的提示，说明 VirtFS 不支持 *mmap* 方法.

5.2.2　多线程文件读写性能

为了评估 StargateFS 多线程文件读写性能，在虚拟机中使用 FIO 工具分别测试 StargateFS 对比 Samba、NFS、VirtFS 共享宿主机中内存文件系统 EXT4-DAX 在 2 个、4 个线程访问时的文件读写性能，每个线程读写的文件大小为 512 MB，2 个线程测试的文件数据总量为 1 GB，4 个线程测试的文件数据总量为 2 GB. 测试 I/O 块大小从 128 KB 增长到 2 MB，共 15 组，测试结果如图 10 所示.

如图 10 中所示，在所有多线程测试用例中，StargateFS 文件读写性能都优于 Samba、NFS 和 VirtFS. 在 2 个线程测试中，StargateFS 平均文件读写性能分别比 Samba、NFS、VirtFS 快 178.3 倍、195.5 倍、61.2 倍. 在 4 个线程测试中，StargateFS 平均文件读写性能分别比 Samba、NFS、VirtFS 快 222.5 倍、256.8 倍、71.5 倍.

测试结果同样突显 StargateFS 使用文件虚拟地址空间直接访问文件和利用硬件 MMU 加速索引文件数据的优势. 另外，StargateFS 多线程顺序读写的性能优于随机读写的性能，如图 10(a)和(c)所示. 在 4 线程读性能测试中，4 线程顺序读最大吞吐量为 25 284.0 MB/s，4 线程随机读最大吞吐量为 23 273.0 MB/s. 原因在于，在顺序读写过程中，一个进程刚访问过的数据可能被另一个进程访问.

实验中还对比 8 台虚拟机同时读写 StargateFS 与 VirtFS 共享宿主机上内存文件系统 EXT4-DAX 中的共享文件的性能. 由于宿主机 CPU 总数有限，每台虚拟机分配 4 个 vCPU. 设置每台虚拟机的系统时间与 Internet 时间同步，在每台虚拟机中使用 Linux 中 Crontab 服务同时开始执行 FIO 单线程测试任务，每个虚拟机中测试文件大小为 512 MB. 测试结果如图 10 所示，EXT4-D_VirtFS(VM8) 表示 8 台虚拟机同时访问 EXT4-D_VirtFS 共享文件读、写特定数据块大小吞吐量的和，StargateFS(VM8)

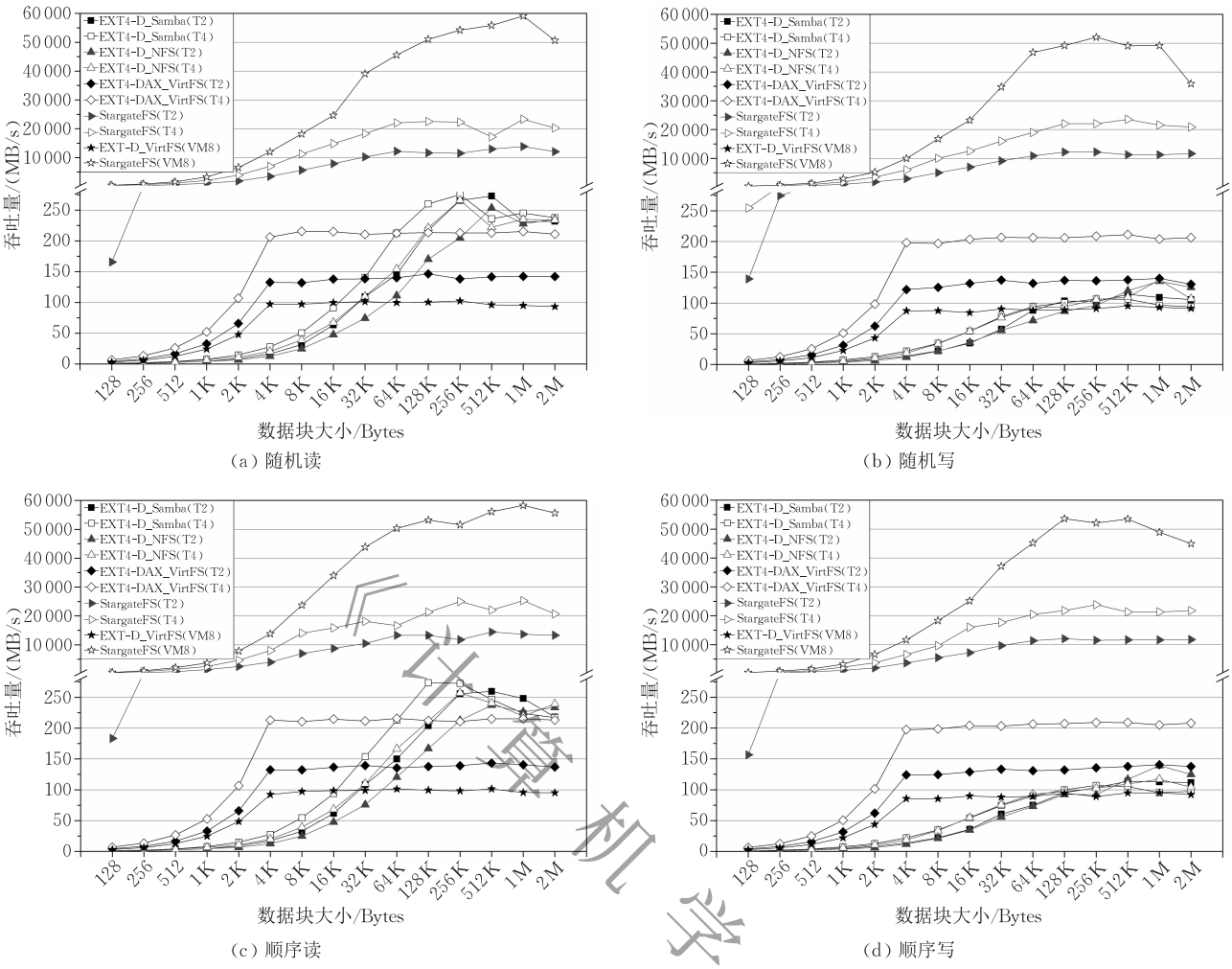


图 10 多进程文件读写性能

表示 8 台虚拟机同时访问 StargateFS 共享文件读、写特定数据块大小吞吐量的和. StargateFS(VM8)平均文件读写性能比 EXT4-D_VirtFS(VM8)快 330.8 倍,8 台虚拟机同时读写共享文件的测试结果同样显示了 StargateFS 使用文件虚拟地址空间和利用硬件 MMU 加速索引共享文件数据页的性能优势.

5.2.3 共享延迟测试

为了比较不同共享文件系统共享文件的延迟,在读者写者应用场景中,通过宿主机用户进程向共享文件中写入一定数据的时间,加上虚拟机用户进程从该共享文件中读出相同数据的时间,来测试共享文件的延迟.需要使用 2 个共享信号量来控制读者、写者读写文件的顺序.宿主机用户进程创建一个新的文件,以读写模式打开,分别以 1 KB、4 KB、1 MB 的 I/O 块大小写入 2 GB 的数据.当宿主机用户进程完成文件写入后,虚拟机用户进程分别以 1 KB、4 KB、1 MB的 I/O 块大小将这 2 GB 的文件数据读取出来,记录宿主机进程写入数据到虚拟机进程读完数据所用的时间.分别测试了 Samba、VirtFS、StargateFS 共享方法的时延,Samba、VirtFS 的配置同 5.2 节,测试结果如表 1 所示.

表 1 共享文件时间 (单位:s)

块大小	Samba	VirtFS	StargateFS
1 KB	843.52	239.94	9.632
4 KB	457.33	66.86	7.067
1 MB	117.73	63.52	5.828

表 1 中的测试结果显示,StargateFS 的平均共享速度分别比 Samba、VirtFS 快 62.9 倍、16.4 倍.有一部分时间开销来源于用户进程多次调用 *write*、*read* 系统调用.测试结果表明,StargateFS 使用共享内存作为存储空间能有效减少文件数据的拷贝次数,通过文件虚拟地址空间利用 MMU 加速访问文件,其文件共享速度对比 Samba、VirtFS 共享方法有很大的提升.

5.3 共享信号量测试

StargateFS 共享内存信号量为不同虚拟机、宿

主机用户进程间同时访问相同共享文件提供同步机制.为了测试共享内存信号量的性能开销,通过在虚拟机上多次申请释放共享信号量的时间来对比不同信号量的性能,实验配置同 5.1 节.StargateFS 分别在用户空间和内核空间实现了共享信号量变量的操作,实验分别测试用户态共享信号量和内核态共享信号量的多次申请释放的时间开销.

5.3.1　用户态共享信号量测试

用户态共享信号量是基于不同虚拟机、宿主机用户进程间的共享内存.实验中对比测试基于 StargateFS 共享内存与基于 Nahanni[32] 共享内存实现的用户态共享信号量申请释放的时间开销,实验测试结果如表 2 中所示.

表 2　用户态共享信号量申请释放时间　(单位:ms)

共享信号量	时间开销
StargateFS 用户态共享信号量	0.308
Nahanni 用户态共享信号量	0.713

表 2 的测试结果表明,基于 StargateFS 共享内存用户态共享信号量申请释放时间比基于 Nahanni 共享内存用户态共享信号量快 2.31 倍.用户态信号量是以多机用户进程间的共享内存为空间存放锁变量,其性能开销主要在于共享内存的建立和释放.Nahanni 建立不同虚拟机、宿主机用户进程间的共享内存包括 3 个步骤:(1) Nahanni 通过 POSIX 共享内存机制在宿主机用户进程与 QEMU 进程(虚拟机)间实现内存共享,在 QEMU 进程(虚拟机)启动时通过 *shm_open* 函数打开共享内存对象,并通过 *mmap* 系统调用把共享内存映射到 QEMU 进程(虚拟机)的虚拟地址空间;(2) 在 QEMU 中添加虚拟 PCI 设备 ivshmem,当虚拟机启动时为虚拟 PCI 设备 ivshmem 分配设备内存;(3) 在虚拟机中通过 UIO① 技术,把虚拟 PCI 设备中的共享内存映射到虚拟机用户进程中.因此,Nahanni 在建立共享内存时,需要打开用于建立共享内存的虚拟 PCI 设备,再把虚拟 PCI 设备中的内存空间映射到用户空间,时间开销大.StargateFS 用户态共享内存直接在虚拟机和宿主机中,使用系统调用在内核空间把预留的内存空间映射到用户空间,不涉及虚拟 PCI 设备的操作,时间开销小.

5.3.2　内核态共享信号量测试

为了评估内核态共享信号量的性能,在虚拟机中分别对内核空间共享信号量和 Linux 中 System V IPC 信号量进行申请释放测试.测试结果如表 3 中所示.

表 3　内核态共享信号量申请释放时间　(单位:μs)

信号量	时间开销
内核空间共享信号量	7.128
System V IPC 信号量	12.405

根据表 3 中的测试结果,内核态共享信号量的申请释放时间比 System V IPC 信号量快 1.74 倍.内核空间共享信号量的物理空间是预先分配的,且固定映射到内核的虚拟地址空间中,申请释放不涉及到空间分配和回收.System V IPC 信号量是用户进程调用 *semget* 系统调用后,在内核中新分配一段空间创建的内核对象.因此,内核空间共享内存信号量的申请释放时间比 System V IPC 信号量快.

6　相关工作

6.1　文件虚拟地址空间

SIMFS[18] 提出了"文件虚拟地址空间"和"文件页表"的概念,利用硬件 MMU 加速文件数据读写,性能达到接近内存总线带宽的速度.StargateFS 借鉴文件页表的概念,提出同时适用在虚拟机中两次地址转换与在宿主机中一次地址转换索引共享文件数据的共享文件页表,完成同驻虚拟机间、虚拟机宿主机间利用硬件 MMU 高效地共享文件.对比 SIMFS 和 StargateFS,主要有以下区别:

(1) SIMFS 是本地内存文件系统,在虚拟机中需通过传统的共享文件系统如 VirtFS、NFS 等才能共享宿主机或其它虚拟机上 SIMFS 中的文件;StargateFS 是同驻虚拟机间、虚拟机宿主机间的共享内存文件系统,直接支持同驻虚拟机间、虚拟机宿主机高效地共享文件;

(2) StargateFS 通过半虚拟化提高虚拟机宿主机共享接口层次的思想,实现了虚拟机间、虚拟机宿主机间的内存共享;SIMFS 不涉及到虚拟机间、虚拟机宿主机间的内存共享;

(3) 为了在虚拟机和宿主机上都能使用硬件 MMU 加速访问共享文件数据,StargateFS 使用共享文件页表组织共享文件数据,解决了在虚拟机中内存寻址两次地址转换和在宿主机上内存寻址一次

① UIO [Online]. https://www.kernel.org/doc/html/v4.12/driver-api/uio-howto.html, 2006.12.11

地址转换访问共享文件数据的问题;而 SIMFS 只能作为虚拟机或宿主机的本地文件系统.所以,当 SIMFS 作为虚拟机的本地文件系统时,其文件页表只能完成虚拟机的虚拟地址到虚拟机的物理地址的转换,而不能完成虚拟机的虚拟地址到虚拟机的物理地址,再到宿主机的物理地址的两次转换.同样,当 SIMFS 作为宿主机的本地文件系统时,其文件页表只能完成宿主机的虚拟地址到宿主机物理地址的转换,而不支持虚拟机中的两次地址转换;

(4) StargateFS 实现了同驻虚拟机间,不同系统内核控制路径同时访问 StargateFS 中共享元数据的同步机制;SIMFS 中实现的是一个系统中多个内核控制路径对共享元数据的同步访问;

(5) StargateFS 解决了同驻虚拟机间,不同虚拟机系统 VFS 层文件索引缓存导致的文件索引信息不一致问题;SIMFS 中未涉及 VFS 缓存信息一致性问题;

(6) StargateFS 提供了共享内存信号量机制,使得不同系统多个用户进程间同步地访问相同共享文件;SIMFS 中未涉及共享信号量.

6.2 虚拟机内存共享技术

在虚拟机同驻的许多应用场景中,都存在不同虚拟机共享宿主机物理内存的可能性[33].一方面,当多个虚拟机运行同一操作系统不同实例或虚拟机间包含相同数据时,有共享宿主机相同数据物理内存的机会.因此,在虚拟机同驻的环境中,内存共享可以有效地节省宿主机物理内存资源.另一方面,在云数据中心,虚拟机间、虚拟机宿主机间可以通过共享宿主机物理内存建立的信道高效地传输数据.因此,内存共享还可以提高虚拟机间、虚拟机宿主机间的通信效率.根据不同的需求,许多研究工作都致力于虚拟化环境下实现内存共享.

在 KVM 平台中,内存共享技术按共享内存实现的方法可以概括为 3 类:第一类是从宿主机端出发,申请一段宿主机内存空间模拟为虚拟 PCI 设备,把虚拟 PCI 设备关联到虚拟机,在虚拟机中通过设备驱动程序把共享内存映射到虚拟机内核空间,甚至映射到虚拟机用户空间,实现虚拟机间、宿主机虚拟机之间的内存共享,如 Nahanni[32];第二类是利用宿主机中 QEMU 程序能够直接访问虚拟机物理地址空间的特性来实现虚拟机间、虚拟机宿主机间的内存共享,如 Socket-outsourcing[34];第三类是从虚拟机端出发,在虚拟机上收集应用程序中需要共享数据的物理内存信息,传递给宿主机.宿主机通过收到的信息在宿主机上定位真正的物理页,并通过 *mmap* 方法映射到宿主机用户进程的虚拟地址空间,实现虚拟机用户进程与宿主机用户进程间的内存共享,如 Zero-copy[35].

以上内存共享技术实现的共享内存对应的物理页是随机分布在宿主机物理地址空间中的,不能安装文件系统到这些共享内存空间中.另外,这些共享内存方法没有考虑页表结构在宿主机上一次地址转换和在虚拟机硬件辅助内存虚拟化两次地址转换的应用.为了实现跨虚拟机和宿主机的高效共享内存文件系统,本文在虚拟机和宿主机中分别划分一段物理地址空间,并把虚拟机中划分的物理内存空间线性映射到宿主机中划分的物理内存空间,实现内存共享,并使得虚拟机两次地址转换和宿主机一次地址转都能访问到宿主机上相同的物理地址空间.在 KVM 虚拟化平台下,不同内存共享技术的对比如表 4 所示.相比其他同驻虚拟机间、虚拟机与宿主机间共享内存方法,划分映射实现内存共享方法的应用场景更广,更易实现.

表 4 同驻虚拟机间共享内存方法比较

	Nahanni	Socket-outsourcing	Zero-copy	划分映射
宿主机	申请内存空间	把共享物理页映射到进程地址空间	把共享物理页映射到进程地址空间	划分固定物理内存,映射到进程地址空间
VMM	创建 PCI 设备,并添加到虚拟机	通过虚拟机物理地址定位共享物理页	实现 vhost 后端驱动模块,与虚拟机 VirtIO 前端通信	修改 KVM 模块,使得共享内存中的 GPA 映射到相同页框号 HPA
虚拟机	新增设备驱动	对虚拟机透明	实现 ViritIO 后端驱动	划分固定物理内存,映射到进程地址空间
虚拟机-虚拟机/虚拟机-宿主机	都支持	都支持	虚拟机-宿主机	都支持

6.3 同 步

同步机制依赖于计算机系统硬件提供的硬件原语,这些硬件原语可供软件使用以实现更复杂的同步机制.在 x86 架构中,通过锁前缀[24]在指令执行

期间对总线加锁的方式来保证指令原子执行.

为了避免传统自旋锁无序竞争导致的公平性问题，在 Linux 内核版本 2.6.25 后引入了排队自旋锁(Ticket Spinlock)①. 排队自旋锁锁结构包含两个属性：锁持有者序号 *owner* 和下一个锁持有者序号 *next*，两个属性的初始值为 0. 当内核线程申请排队自旋锁时，先拷贝一个锁结构，原子地将排队自旋锁 *next* 值加 1，并返回原值作为拷贝锁结构的 *owner* 值. 接下来循环比较拷贝锁结构中的 *owner* 值和排队自旋锁的 *owner* 值是否相等，相等则获得锁. 当内核线程释放排队自旋锁时，原子地将排队自旋锁 *owner* 值加 1. 在 x86 架构下，对排队自旋锁 *owner* 值和 *next* 值加 1 的操作是通过加锁前缀的 xaddw 原子操作指令完成的.

在多个虚拟机同驻的虚拟化环境中，把排队自旋锁变量存放在共享内存区域. 不同虚拟机及宿主机内核线程在竞争共享排队自旋锁时拷贝排队自旋锁结构，并通过加锁前缀的原子操作指令 xaddw 修改自旋锁变量的值，实现虚拟机及宿主机内核线程间的同步.

System V IPC 通过系统调用操作内核对象实现进程间的同步，内核对象提供了共享资源的状态和原子操作. 经研究发现[36]，很多同步是无竞争的，同步过程中不必要的系统调用导致了大量的性能开销. Glibc② 实现的 Pthread 多线程库使得锁变量的大部分操作在用户空间完成，只有少部分需要内核支持的操作才通过系统调用陷入内核完成. 属于不同虚拟机、宿主机的进程间同步时，可以通过把锁变量存放在虚拟机间、虚拟机宿主机间共享的内存中，根据不同应用需求，分别在内核空间和用户空间实现对共享锁变量的操作.

7　总　结

本文在虚拟化环境中，利用新型非易失性内存及内存文件系统的优点，面向同驻虚拟机提出了一种新的基于文件虚拟地址空间的共享内存文件系统设计，实现虚拟机之间、虚拟机宿主机之间高效的文件共享. 该设计分别在虚拟机和宿主机中预留了一段虚拟地址空间和物理地址空间. 虚拟机中预留的物理地址空间线性映射到宿主机中预留的物理地址空间，实现虚拟机之间、虚拟机宿主机之间的内存共享. 共享文件通过共享文件页表结构组织文件数据物理页. 在虚拟机和宿主机中分别使用预留的虚拟地址空间访问文件. 打开文件时，把文件的共享文件页表插入到虚拟机和宿主机中进程的页表中，实现文件共享，并利用硬件 MMU 加速定位文件数据物理页，提供了高效的文件读写性能.

基于提出的同驻虚拟机共享内存文件系统设计，在 Linux 系统中基于 KVM 平台实现了一个高效的共享内存文件系统 StargateFS，并基于共享内存实现共享内存信号量，允许不同系统的多个进程间同步地访问相同文件. 实验测试结果表明，StargateFS 的平均文件读写性能比目前性能最优的共享文件系统 VirtFS 快 64 倍，较传统共享文件系统 Samba、NFS 分别提高了 172 倍、191 倍.

参　考　文　献

[1] Liu S, Jia W. A survey: Main virtualization methods and key virtualization technologies of CPU and memory. Open Cybernetics & Systemics Journal, 2015, 9(1): 350-358

[2] Ruest N, Ruest D. Virtualization, a Beginner's guide. Journal of Biological Education, 2009, 32(1): 41-47

[3] Macdonell C, Lu P. Pragmatics of virtual machines for high-performance computing: A quantitative study of basic overheads//Proceeding of the High Perf Computing & Simulation Conference. Prague, Czech Republic, 2007: 704-711

[4] Wang L, Von Laszewski G, Younge A, et al. Cloud computing: A perspective study. New Generation Computing, 2010, 28(2): 137-146

[5] Xu F, Liu F, Jin H, et al. Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions. Proceedings of the IEEE, 2013, 102(1): 11-31

[6] Ibrahim S, Jin H, Cheng B, et al. Cloudlet: Towards map reduce implementation on virtual machines//Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing. Garching, Germany, 2009: 65-66

[7] Foster I, Kesselman C, Nick J M, et al. Grid services for distributed system integration. Computer, 2002, 35(6): 37-46

[8] Ahmad F, Chakradhar S T, Raghunathan A, et al. Shuffle watcher: Shuffle. aware scheduling in multi-tenant MapReduce

① FIFO ticket spinlocks [Online]. http://lkml.org/lkml/2007/11/1/125, 2007.11.1

② The community wiki for GLIBC [Online]. https://sourceware.org/glibc/wiki/HomePage, 2018.3.22

clusters* //Proceedings of the USENIX Annual Technical Conference. Philadelphia, USA, 2014: 1-12

[9] Zhang Q, Liu L. Shared memory optimization in virtualized cloud//Proceedings of the International Conference on Cloud Computing. New York, USA, 2015: 261-268

[10] van Vugt S. File Sharing: NFS, FTP, and Samba. Berkeley, USA: Apress, 2014

[11] Leach P J, Naik D. A common Internet file system (CIFS/1.0) protocol. California, USA: Internet-Draft, IETF, 1997

[12] Pawlowski B, Noveck D, Robinson D, et al. The NFS version 4 protocol//Proceedings of the 2nd International System Administration and Networking Conference. Maastricht, Netherlands, 2000: 1-20

[13] Pfaff B, Garfinkel T, Rosenblum M. Virtualization aware file systems: Getting beyond the limitations of virtual disks//Proceedings of the Conference on Networked Systems Design & Implementation. San Jose, USA, 2006: 353-366

[14] Jujjuri V, Van Hensbergen E, Liguori A, et al. VirtFS—A virtualization aware file system pass-through//Proceedings of the Ottawa Linux Symposium (OLS). Ottawa, Canada, 2010: 109-120

[15] Russell R. virtio: Towards a de-facto standard for virtual I/O devices. ACM SIGOPS Operating Systems Review, 2008, 42(5): 95-103

[16] Mittal S, Vetter J. A survey of software techniques for using non-volatile memories for storage and main memory systems. IEEE Transactions on Parallel & Distributed Systems, 2016, 27(5): 1537-1550

[17] Shen Z, Xue W, Shu J. Research on new non-volatile storage. Journal of Computer Research & Development, 2014, 51(2): 445-453

[18] Sha H M, Chen X, Zhuge Q, et al. A new design of in-memory file system based on file virtual address framework. IEEE Transactions on Computers, 2016, 65(10): 2959-2972

[19] Xu J, Swanson S. Nova: A log-structured file system for hybrid volatile/non-volatile main memories//Proceedings of the 14th USENIX Conference on File and Storage Technologies. Berkeley, USA, 2016: 323-338

[20] Ou J, Shu J, Lu Y. A high performance file system for non-volatile main memory//Proceedings of the Eleventh European Conference on Computer Systems. London, UK, 2016: 12-28

[21] Zhang Xue-Cheng, Xiao Nong, Liu Fang, et al. Summary of in-memory file system. Journal of Computer Research and Development, 2015(S2): 9-17(in Chinese)
(张学成, 肖侬, 刘芳等. 内存文件系统综述. 计算机研究与发展, 2015(S2): 9-17)

[22] Liu J. Evaluating standard-based self-virtualizing devices: A performance study on 10 GbE NICs with SR-IOV support//Proceedings of the IEEE International Symposium on Parallel & Distributed Processing. Atlanta, USA, 2010: 1-12

[23] Habib I. Virtualization with KVM. Houston, USA: Belltown Media, 2008

[24] Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual, Combined Volumes: 1, 2A, 2B, 2C, 3A, 3B and 3C. System Programming Guide, 2012: 32-2

[25] Bovet, Daniel, Cesati, et al. Understanding the Linux Kernel, 3rd Edition. New York, USA: O'Reilly & Associates, 2007

[26] Mauerer W. Professional Linux Kernel Architecture. New Jersey, USA: John Wiley & Sons, 2010

[27] Drepper U. Futexes Are Tricky. Tokyo, Japan: Red Hat Inc., 2005

[28] Dollimore J, Kindberg T, Coulouris G. Distributed Systems: Concepts and Design. New Jersey, USA: Addison-Wesley, 2005

[29] Butenhof D R. Programming with POSIX Threads. New Jersey, USA: Addison-Wesley, 1997

[30] Culler D E, Singh J P, Gupta A. Parallel computer architecture: A hardware/software approach. Amsterdam, Netherlands: Gulf Professional Publishing, 1999

[31] Dulloor S R, Kumar S, Keshavamurthy A, et al. System software for persistent memory//Proceedings of the Ninth European Conference on Computer Systems. Amsterdam, Netherlands, 2014: 15

[32] Macdonell A C. Shared-memory optimizations for virtual machines. Edmonton, Canada: University of Alberta, 2011

[33] Gupta D, Lee S, Vrable M, et al. Difference engine: Harnessing memory redundancy in virtual machines. Communications of the ACM, 2010, 53(10): 85-93

[34] Eiraku H, Shinjo Y, Pu C, et al. Fast networking with socket-outsourcing in hosted virtual machine environments//Proceedings of the ACM Symposium on Applied Computing. Honolulu, USA, 2009: 310-317

[35] Pinto C, Reynal B, Nikolaev N, et al. A zero-copy shared memory framework for host-guest data sharing in KVM//Proceedings of the Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress. Toulouse, France, 2016: 603-610

[36] Franke H, Russell R, Kirkwood M. Fuss, Futexes and Furwocks: Fast userlevel locking in Linux//Proceedings of the Australian Unix systems User Group (AUUG). Ontario, Canada, 2002, 85-104

Edwin H-M SHA, born in 1964, Ph.D., professor, Ph.D. supervisor. His research interests include new-generation memory systems, cloud computing, operating systems, embedded systems and software, and communication security.

WU Ting, born in 1991, Ph. D. candidate. His research interests include system virtualization and in-memory file system.

ZHUGE Qing-Feng, born in 1970, Ph. D., professor, Ph. D. supervisor. Her research interests include operating systems, embedded systems and software, and optimization algorithms.

YANG Chao-Shu, born in 1984, Ph. D. candidate. His research interests focus on persistent memory file system.

MA Zhu-Lin, born in 1993, Ph. D. candidate. Her research interests focus on main memory database.

CHEN Xian-Zhang, born in 1989, Ph. D., lecturer. His research interests include new-generation memory systems, file systems, embedded systems and software, and cloud computing.

Background

Virtual machine is a fundamental infrastructure of cloud computing. The performance of virtual machines is critical for user experience. Shared file system is an efficient technology for improving the performance of virtual machines located on the same physical machine (i. e., co-resident virtual machines). On the other hand, emerging persistent memories, such as 3D XPoint, show extremely high performance and non-volatility, which make persistent memories as the promising storage for shared file data.

However, existing shared file systems for co-resident virtual machines rely on either virtualized network device (e. g., Samba) or virtualized I/O path (e. g., VirtFS), which cannot fully exploit the benefits of persistent memory.

In this paper, we propose a novel shared file system installed on the shared persistent memory. First, the shared files can be exposed to multiple co-resident virtual machines via a shared page table. Second, the software stacks and data copies of the I/Os for shared files are reduced by directly accessing files using virtual address space and hardware MMU rather than the network. Third, to ensure the consistency and concurrency of file accesses, this paper presents a new synchronous mechanism for shared data, such as file data and metadata. This paper also implemented a real shared memory file system, StargateFS in widely used KVM platform. As a result, the average throughput of StargateFS achieves 64 times, 172 times and 191 times higher than VirtFS, the state-of-the-art shared file system, Samba, and NFS, respectively.

This paper is supported by the National High Technology Research and Development Program (863 Program) of China under Grant No. 2015AA015304, the National Natural Science Foundation of China under Grant No. 61472052, the Chinese Postdoctoral Science Foundation under Grant No. 2017M620412. These projects aim to optimize big data systems with emerging persistent memories. The authors have developed several persistent memory file systems and optimization algorithms. Many papers have been published in international conferences and journals. This paper is part of the topic for optimizing cloud computing with persistent memory.