高技术通讯 2021 年 第31 卷 第11 期:1117-1125

doi:10.3772/j.issn.1002-0470.2021.11.001

MIPS VZ 虚拟化环境中透明大页性能的优化①

朱 琛②*,**,***,**** 王 剑③*,**,*** 高 翔**** 毛碧波**** 杨小娟****

(*计算机体系结构国家重点实验室(中国科学院计算技术研究所) 北京 100190)

(**中国科学院计算技术研究所 北京 100190)

(***中国科学院大学 北京 100049)

(****龙芯中科技术有限公司 北京 100190)

摘 要 内存地址翻译一直是影响计算机性能的重要因素之一,在虚拟化环境中由于存在多级地址翻译,这一问题更为突出。使用透明大页(THP)是优化这一问题的常见手段。本文以龙芯 3A4000 处理器和 KVM 虚拟机为实验平台,分析了 MIPS VZ 环境中虚拟机使用透明大页的性能。发现透明大页在应用中存在分组 TLB 负载不均衡,使得部分场景中透明大页对性能产生了负面影响。本文对现有 KVM 内存管理进行了改进。通过在内存管理中增加对分组 TLB 负载的考量,提升了 KVM 虚拟机使用透明大页的性能。该方法对其他使用 TLB 分组结构的处理器也有参考价值。

关键词 透明大页(THP);虚拟化;MIPS;内存管理

0 引 言

内存管理是影响计算机性能的重要因素之一,操作系统的虚实地址转换依赖于 CPU 的内存管理单元(memory management unit,MMU),在访存密集型应用中 MMU 重填地址转换缓冲器(translation lookaside buffer,TLB)的开销容易成为性能瓶颈[1-2]。在硬件辅助虚拟化环境中,为了减少虚拟机内存缺页的次数,往往采用两级 TLB,一个虚拟机的虚拟地址转换需要两个 TLB 表项,这进一步加重了该问题。采用内存大页技术可以明显增大 TLB 对内存的覆盖率,通常可以减少 TLB 的缺页次数,因此常被用于优化虚拟机访存[3]。现有很大一部分处理器采用分组 TLB 实现对大小页的支持。透明大页分配机制在面对使用大容量连续数据集、访存频繁且局部性差的程序时会造成这类处理器的 TLB 负载不均衡,影响系统性能。本文针对这一问题提出了一种优化 TLB 负载均衡度的方法,并创新地提出了一种高效的 TLB 负载均衡度监测机制和基于负载不均衡度的大页调度策略。

本文的内容如下,第 1 节以 MIPS 处理器为例介绍了 MMU 中内存硬件辅助虚拟化的方法和 MMU 对大页的支持。第 2 节以包含虚拟化组件(virtualization module,VZ)扩展的龙芯 3A4000 处理器为平台,通过实验对比了 MIPS 处理器在虚拟化环境下启动大页前后 SPEC CPU 2006 测试中的性能。第3 节对 SPEC CPU 2006 测试中性能异常的项目进行了分析。第 4 节对现有的内存分页策略进行了改进。第 5 节对改进后的 KVM 虚拟机进行了性能测试。第 6 节总结了全文并展望了未来工作。

1 背 景

1.1 虚拟机的地址翻译

内存页是操作系统分配内存的最小单位,为了

① 中国科学院战略性先导科技专项(C 类)课题(XDC05020100)资助项目。

② 男,1983 年生,博士生;研究方向:计算机系统结构;E-mail:zhuchen@ict.ac.cn。

③ 通信作者,E-mail:jw@ict.ac.cn。

(收稿日期:2020-11-20)

保护不同进程的数据,用户进程使用虚拟地址,MMU 查询 TLB,将虚拟地址翻译为物理地址进行访问。如果 TLB 中未命中,页表遍历器(pagewalk)会查询页表并将对应的表项填入 TLB 中。

虚拟化需要使用两级页表完成虚拟机的地址翻译,一级是虚拟机维护的客模式虚拟地址(guest virtual address,GVA)到客模式物理地址(guest physcial address,GPA)的页表。一级是虚拟机管理器(virtual machine monitor,VMM),本文使用 KVM 作为 VMM 维护 GPA 到宿主机物理地址(host physical address,HPA)的翻译[4]。除此之外,虚拟机的虚拟外设也会使用正常的宿主机虚拟机地址(host virtual address,HVA)到 HPA 的页表项。

虚拟机地址翻译依赖于 MMU 和 TLB 的硬件辅助。常见的虚拟机内存地址翻译的硬件辅助根据使用 TLB 的层级可分为两种。

第一种方式是虚拟机使用一级 TLB 完成虚拟机的地址翻译,虚拟机的 TLB 表项中只记录 GVA 或 GPA 到 HPA 的转化。所有的 TLB 操作由宿主机内核态完成[5]。早期的 X86 处理器、龙芯的 GS464E 处理器核使用的虚拟化扩展采用这种方式。一次虚拟机 TLB 重填的流程如图 1 所示。

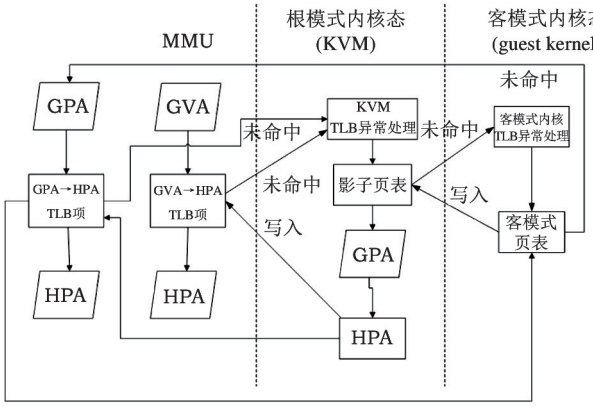

图 1 虚拟机地址翻译流程

这种方式下,每次 TLB 重填需要先产生宿主机例外,查询 KVM 中的 GVA→GPA 影子页表,如影子页表未命中需要返回客模式通过 GVA 查询 GPA,最终再回到 VMM 通过 GPA 找到 HPA,并将 GVA 到 HPA 的对应关系填入页表。这种方式的优点是硬件实现较为简单,占用 TLB 表项少,只需对 TLB 和 MMU 做少量更改就可实现虚拟机的内存虚拟化;缺点也很明显,一次页表的填入经常需要在客模式和根模式下多次切换,每次切换 CPU 都需要保存大量的状态信息,开销较大。

另一种方式是使用两级 TLB 完成虚拟机地址的翻译,其中一级记录 GVA 到 GPA 的转化,这一级 TLB 的操作在客模式内核态完成。第二级记录 GPA 到 HPA 的转化,这一级的 TLB 操作由宿主机内核态完成[6]。Intel 的扩展页表(extended page tables,EPT)、AMD 的嵌套页表(nested page tables,NPT)和 MIPS 的 VZ 都使用这一方式。MIPS VZ 中的地址翻译逻辑如图 2 所示。

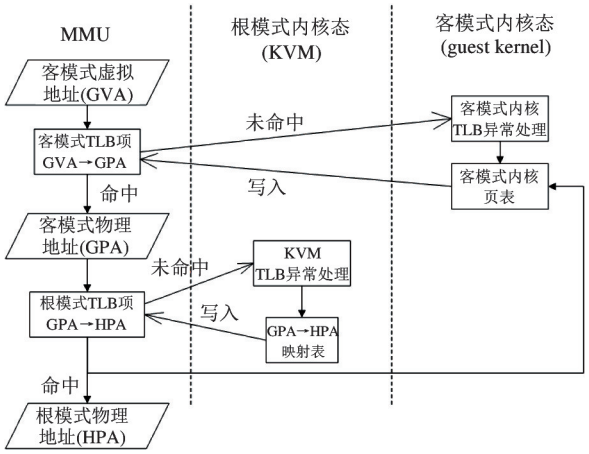

图 2 VZ 虚拟机地址翻译流程

在这种模式下,pagewalk 分别查询虚拟机和 VMM 的页表完成各自的 TLB 重填操作。两个表项一个记录 GVA→GPA,一个记录 GPA→HPA,由 MMU 根据两个表项完成地址的翻译。这样在一定程度上避免了多次切换 CPU 状态的问题。除此之外,由于 GVA→GPA、GPA→HPA 的映射是完全相互独立的,两级地址翻译之间实现了解耦,方便使用透明大页等内核的内存优化技术。

使用两级 TLB 后虚拟机环境地址翻译的代价仍然明显高于非虚拟机环境。在客模式内核态查找 GVA→GPA 的过程中,需要查询虚拟机的各级页表,如果页表所在 GPA 对应的 GPA→HPA 表项不在 TLB 中时,会产生额外的 VMM 缺页异常,返回 VMM 填入对应的 GPA→HPA 页表项,每访问一级客模式页表都可能产生额外的重填,每次额外重填

会带来 1 次客模式到根模式的切换和数次访存。一次重填最多会产生 $M \times N + M + N$ 次访存,其中 M 和 N 分别为宿主机和虚拟机的页表级数[7]。如图 3 所示,如果 M 和 N 都为 4,一次重填最多需要 24 次访存。

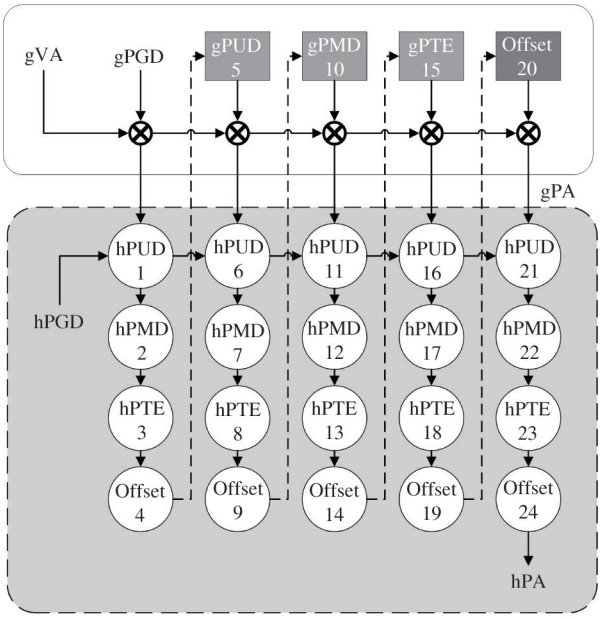

图 3　虚拟机两级页表的遍历过程

1.2　操作系统和处理器对大页的支持

操作系统在多级页表中习惯于用一个内存页存储一个目录级,以 4 kB 页为例,在 64 位地址的条件下,一个 4 kB 页可以储存 512 个 64 位地址,即页表项(page table entry, PTE)的上一级页中间目录(page middle directory,PMD)可以覆盖 512 ×4 kB = 2 MB 的地址空间,上一级的页上级目录(page upper directory,PUD)为 512 ×2 MB 即1 GB,为了管理方便,一般一个大页直接对应其中一个目录项。如 X86-64 使用 4 kB 小页对应的大页即为 2 MB 和 1 GB。

MMU 根据设计可以支持一种或几种不同规格的内存页大小,对应的 TLB 也支持相应的页大小。较小的页大小能够避免内存的浪费,但由于 TLB 的数量限制,能够覆盖的内存范围有限,因此在一些特定的应用场景,采用更大的内存页(即大页)能够有效提升 TLB 的内存覆盖率,提升性能。

MIPS 处理器一般有 4 种不同的 TLB,分别是指令 TLB(ITLB)、数据 TLB(DTLB)、固定页大小 TLB(FTLB)和可变页大小 TLB(VTLB)。ITLB 和 DTLB 统称为微 TLB(μTLB),FTLB 和 VTLB 统称为联合 TLB(JTLB)。μTLB 对软件透明,由硬件从 JTLB 中查询后填入,因此其内容是 JTLB 的子集[8]。JTLB 中 FTLB 同时只能使用一种页大小,VTLB 同时支持不同页大小(步进与 μTLB 相同)。VTLB 使用全相连结构支持不同页大小,由于结构限制其项数较少。以龙芯 3A4000 使用的 GS464V 处理器核为例,ITLB 和 VTLB 为 64 项,DTLB 为 32 项,FTLB 为 2048 项。

在 VZ 模式下,JTLB 中有 GVA→GPA、GPA→HPA 2 种表项,由软件页表遍历器填入,MMU 的硬件逻辑将 2 种表项翻译成 GVA→HPA 的单一表项填入 μTLB。JTLB 中的 FTLB 项数较多,一般用于小页,VTLB 一般用于大页。其他架构的处理器,如 X86 和 Sparc 的部分处理器也使用了大小页 TLB 分组的结构[9-10]。

2　VZ 虚拟化环境下的大页性能测试

Linux 系统中对大页的应用主要分为 2 种模式,即大页文件系统(Hugetlbfs)和透明大页(transparent huge pages,THP)。Hugetlbfs 需要操作系统手动预先分配大页并映射为一个文件,用户程序可以通过该文件获取按大页预分配的内存。THP 是一种用户透明的大页优化,内核会根据进程的内存使用情况,自动分配大页或将内存中的小页合并成大页。THP 避免了 Hugetlbfs 需要应用程序显式调用的问题,使大页的应用更为广泛[11]。

MMU 使用大页需要被翻译的地址和翻译的地址同时满足大页的对齐要求,KVM 使用的内存是在根模式用户态申请的,根内核 THP 的分配规则是将 HVA 和 HPA 按大页对齐,却没有维护 HPA 和 GPA 的对齐关系。如果 HVA、HPA 和 GPA 三者不能同时满足大页对齐时,即使根文件系统和虚拟机都使用了透明大页,最终 KVM 在 TLB 中填入的表项仍是小页。如图 4 所示,HVA 的013FFC000 对应 GPA 的 0xA000000,虽然根模式和虚拟机内核都使用了大页,最终 TLB 填入的仍是小页。该问题解决办法

是在用户态启动时考虑对齐问题,申请虚拟机内存对应的 HVA 地址时额外增加一个大页大小的虚拟地址空间,取第一个大页对齐的 HVA 地址为客户机 GPA 的起始地址。

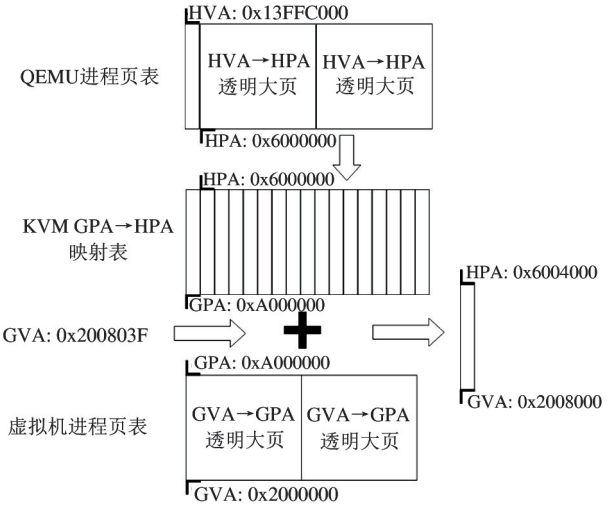

图 4 HVA 和 GPA 不对齐时页表的状态

测试中虚拟机和 KVM 都通过 THP 的方式使用大页,使用 SPEC CPU 2006 测试作为对比基准。CPU 为龙芯 3A4000 处理器,主频 1.8 GHz。物理机和虚拟机都设置为双核,内存均为 4 GB,测试中未开启 3A4000 的向量优化。测试中涉及 4 种组合:物理机不开启透明大页,物理机开启透明大页,虚拟机和 KVM 不开启透明大页,虚拟机和 KVM 开启透明大页,测试成绩对比如图 5 所示。

可见使用大页后,SPEC CPU 2006 中 GemsFDTD、cactusADM、bwaves、xalancbmk、astar、omnetpp、sjeng 都有较大幅度的性能提升,且宿主机和虚拟机使用大页前后的性能变化趋势一致。但 mcf 程序的成绩出现了异常,物理机使用大页提升了 16.2%,但虚拟机使用大页却下降了 9.2%。除此之外的其他测试项目,大页的影响较小。

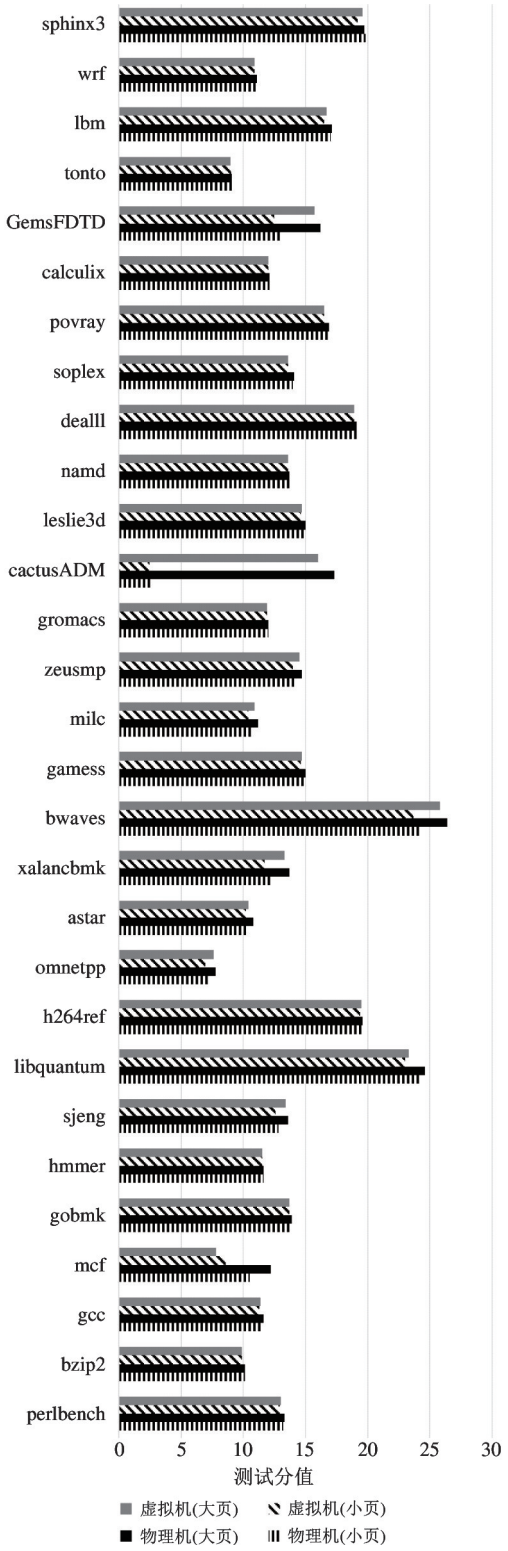

图 5 龙芯 3A4000 在不同组合情况下的 SPEC2006 CPU 性能对比

3 测试项性能下降的原因分析

现有研究认为透明大页的性能问题主要来源于以下几个方面:(1)内存碎片化和多虚拟机内存超分造成的大页分配困难[11];(2)透明大页分配、合并和拆分的开销造成的性能抖动;(3)多个虚拟机或进程频繁切换引起的 TLB 失效[12]。针对上述问题作了如下实验,宿主机只启动一台虚拟机且没有其

他活跃进程。虚拟机启动时使用 Hugetlbfs 预分配大页,确保虚拟机在 GPA→HPA 转换时一定是大页,且虚拟机使用时没有大页分配、合并的相关开销。同时监测 mcf 测试运行时进程的大页使用情况,确定 mcf 运行时成功分配了大页。这种情况下,mcf 的测试成绩为 7.84,与使用两级透明大页时的测试分值 7.79 相差仅为 0.6%,因此可以排除是因为大页分配失败引起的性能下降。在该实验的基础上修改了 TLB 重填例外代码,对 TLB 的重填来源进行了统计,结果显示测试时虚拟机 99% 以上的重填都来自于 mcf 测试进程,宿主机 99% 以上的重填都来自于运行 mcf 的虚拟机,排除了多个虚拟机或进程频繁切换引发的 TLB 失效。最后通过 perf 统计了虚拟机透明大页分配、合并和拆分相关函数的时间占比,统计结果显示透明大页相关函数的时间占比小于总时间的 0.1%。因此现有研究不能够解释 mcf 测试在龙芯 3A4000 虚拟机使用透明大页时性能下降的原因。

为了进一步分析 mcf,在图 5 的组合基础上,增加了关闭 KVM 透明大页保留客模式透明大页和保留 KVM 透明大页关闭客模式透明大页 2 种情况的测试。测试结果如图 6 所示。测试结果中可以看到,无论是 GVA→GPA 还是 GPA→HPA 使用大页对性能的影响都是正向的。但同时对 GVA→GPA 和 GPA→HPA 使用大页时,却产生了明显的负面影响。

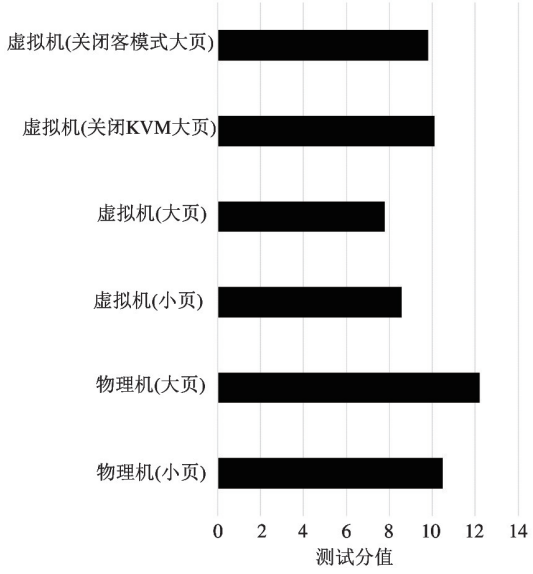

图 6　不同大页策略下的 mcf 分值对比

mcf 是一个 C 语言实现的,用于大型公共交通中的单站车辆调度的程序,实测在 64 位系统中占用约 1.6 GB 的连续内存地址,访存频繁且访存局部性差。测试系统默认使用 16 kB 小页,对应的 PMD 尺寸为 32 MB。GS464V 用于大页的 TLB 为 VTLB,共有 64 项。在物理机运行时,64 项大页可覆盖 2 GB 的地址空间。在虚拟机模式下运行时,由于使用两级页表,需要 VTLB 中同时用 2 个页表项才能完成一次地址翻译,因此最大覆盖的不需要替换的地址范围为 1 GB。这意味着 GS464V 在默认情况下,使用大页能覆盖 mcf 测试的全部内存地址,在虚拟机模式下则不能全部覆盖。这在 mcf 程序中会带来频繁的 TLB 重填影响性能。为了进一步验证,在 Linux 内核通用(物理机和虚拟机复用,负责填入 HVA→HPA 或 GVA→GPA 的页表项)和 KVM 模块专用(负责填入 GPA→HPA 的页表项)的 TLB 重填例外入口加入了计数,并区分大小页,统计不同条件下测试 mcf 每秒钟 TLB 重填的次数。测试分值和重填次数见图 7。图中曲线对应测试分值,左边的纵坐标标示每秒钟重填次数,右边的纵坐标标示测试分值。从左到右测试环境依次为宿主机关闭透明大页、宿主机开启透明大页,虚拟机环境关闭所有透明大页,虚拟化环境只使用虚拟机透明大页和虚拟机环境开启两级透明大页。

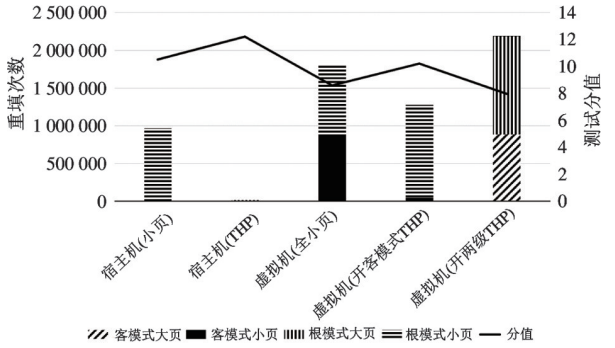

图 7　不同环境下 mcf 测试的 TLB 重填次数和分值

可见测试分值与重填次数呈明显相关性,重填次数越多,测试分值越低。使用透明大页时,Linux 的内存管理器倾向于尽可能分配大页[11],因此在两级内存管理都开启 THP 时,所有的页面都使用大页,TLB 的重填主要局限于大页的 VLTB,达

218 万次/s,而 FTLB 仅重填不到 450 次/s。手动关闭 KVM 的 THP 后总重填次数下降到 128 万次。这表明 2 种 TLB 之间的负载不均衡,资源没有得到充分利用。

4 基于 TLB 负载均衡的虚拟机性能优化

由上述分析可知,现有的 Linux 透明大页分配机制会造成面对使用大容量连续数据集、访存频繁且局部性差的应用程序时,当大页的表项不足以覆盖热点内存区域会产生对 TLB 表项的剧烈竞争,造成应用程序使用大页后性能下降。硬件上解决这一问题最直接的办法是增加表项数量,但 TLB 的数量增加受时序限制,也会增加硬件成本。与此同时用于小页的 TLB 表项在 mcf 测试中处于空闲状态没有充分利用。使用单级大页后性能提升的实验结果表明,如果充分利用空闲的 TLB 资源,可以减少 TLB 替换次数,提升性能。

优化该问题需要通过对 TLB 的重填状况进行监测,识别出不平衡状态。现有的相关性能计数器只提供了总的 TLB 重填次数,但 MIPS 的 TLB 软件重填机制和核内空闲寄存器为高效统计大页重填的次数提供了可能。通过大页重填次数和总的重填次数,就能够得到 TLB 的负载不均衡度。在物理机上一般同时运行数百个到上万个进程,对之进行监控代价较高。同时由于物理机运行的负载较为复杂,行为模式经常发生变化。同时在一个物理机上运行的虚拟机数目有限,且虚拟机通常长时间运行稳定负载,因此在 KVM 中对虚拟机进行监控和优化更为合理。利用现有硬件资源在 KVM 中设计一个高效的 TLB 负载检测机制,并根据 TLB 负载不均衡度调整大页分配比例,可以提高 TLB 的命中率,提升系统性能。

4.1 TLB 负载检测机制

TLB 重填是影响 CPU 性能的关键路径,因此需要一个高效的 TLB 负载检测机制,以避免对系统产生负面影响。本文中使用的检测方法,充分利用龙芯 GS464V 处理器核的硬件资源,实现了可开关的软件大页重填计数器,避免了在 TLB refill 中增加访存,具体结构如图 8 所示。

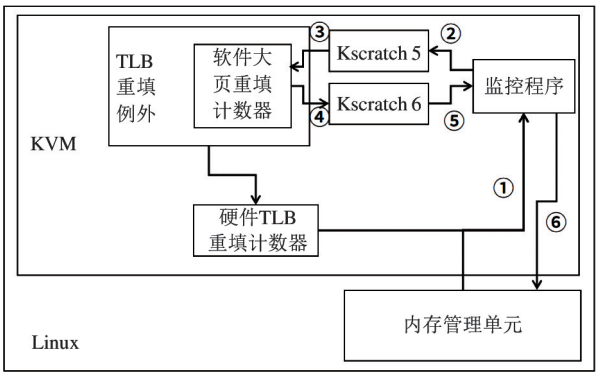

图 8 TLB 负载检测机制

该检测方法利用了龙芯 3A4000 处理器中的以下硬件资源:硬件 TLB 重填计数器(以下简称 TLB 计数器)和 2 个空闲的 Kscratch 寄存器。

硬件 TLB 重填计数器是 GS464V 中的一个性能计数器,能够统计总的宿主机 TLB 重填次数,但无法区分大小页。Kscratch 寄存器是 GS464V 中的一组 64 位可读写寄存器,用于存放核心态软件的暂存数据。GS464V 中一共有 6 个 Kscratch 寄存器,为 Kscratch 1 ~ 6。在现有的 Linux 内核中,除了 KSratch 1,其余 5 个没有使用。本文使用了 Kscratch 5 和 Kscratch 6 两个寄存器。其中 Kscratch 5 用于开关软件大页重填计数器,Kscratch 6 用于软件大页重填计数器存放计数值。

软件上增加 2 个模块,TLB 负载监控程序和软件大页重填计数器(以下简称大页计数器)。TLB 负载监测程序在时钟中断例外中执行,软件大页重填计数器位于 TLB 重填例外函数中。除此之外对内核的内存管理单元进行了少量修改,增加了限制虚拟机大页分配数量的逻辑。该逻辑中使用了 2 个参数:当前虚拟机允许分配的大页数 N_{hl} 和当前虚拟机使用的大页数 N_{hc}。N_{hl} 的初始值设定为虚拟机内存大小除以系统大页尺寸,即虚拟机所有的内存都可使用大页。N_{hc} 的值由内存管理单元维护,记录当前虚拟机使用的大页数。

具体流程如图 8 所示。

(1)TLB负载监控程序通过读取硬件TLB重填

计数器和 N_{hc}、N_{hl},统计当前 MMU 的负载和大页使用情况。

(2)当监控程序判别有必要监测大页重填计数时,向 Kscratch 5 寄存器写入非零值。

(3)TLB 重填时读取 Kscratch 5 寄存器,当值为非零时,开启软件大页计数器。

(4)当前填入的页为大页时,软件计数器更新 Kscratch 6 寄存器的计数值。

(5)监控程序根据实时的总重填计数和大页重填计数,判断是否处于负载不均衡的状态。

(6)当监控程序判别不均衡需要调整透明大页分配策略时,步进式调整 N_{hl}。

4.2　基于负载不均衡度的大页调度策略

监控程序通过 TLB 重填频率,结合处理器单次 TLB 重填耗时,可以判断 TLB 重填在 CPU 运行中的耗时占比。当耗时占比高于阈值时,并不会马上启动后续流程,而是会更新一个计数器(以下称该计数器为 C_L)。当连续高于阈值时 C_L 会逐次累加,而中间有不满足阈值的情况时 C_L 会逐次递减直至为 0,当 C_L 累加到一定数量时,监控程序才会启动大页计数器。本文将启动条件设为 $C_L \geqslant 2560$,这意味着 TLB 连续处于高负载 10 s 以上。这样做是为了避免调度算法过于敏感,频繁切换状态造成系统性能扰动。

启动大页计数器后,获取一个时钟中断间隔内的大页重填次数 N_{hr} 和总的重填次数 N_r。由于大页和小页的 TLB 项数不同,用 N_{hr} 和 $N_r - N_{hr}$ 分别除以大页和小页的 TLB 项数,得到平均每个大页 TLB 表项重填的频率 f_h 和平均每个小页 TLB 表项的重填频率 f_s,不平衡程度 y 通过下列函数衡量:

$$y = \begin{cases} \dfrac{f_h}{f_s} & f_h \geq f_s \\ \dfrac{f_s}{f_h} & f_s > f_h \end{cases}$$

y 连续计算 3 次,求取其算术平均值。当不平衡度大于阈值时,启动大页调度程序,本文实验使用的不平衡度阈值为 20。

在 y 超出阈值且 $f_h > f_s$ 时,表明负载主要集中于大页使用的 VTLB。监控程序将 N_{hl} 更新为 $N_{hc} - 1$。此时,Linux 内存管理单元检测到 $N_{hc} > N_{hl}$,会拆分大页至 $N_{hl} = N_{hc}$。如果此时 f_h 和 f_s 大小关系没有发生翻转且 y 仍高于阈值,会进一步减小 N_{hl},直至 f_h 和 f_s 大小关系发生翻转或 y 降低到阈值以下。此后监控程序会关闭大页计数器,并将 C_L 清零。

如果负载集中于 FTLB 且 $N_{hc} < N_{hl}$,表明虚拟机此时需要等待内存管理程序合并透明大页,且内存管理程序本身具有权限,此时监控程序会关闭大页计数器,并将 C_L 清零。如果 $N_{hc} = N_{hl}$,且 N_{hl} 覆盖范围比虚拟机物理内存小,将扩大 N_{hl}。

5　优化后的性能测试

为了验证改进的效果,优化后重新测试了 mcf,并对 TLB 重填次数进行了重新统计,结果如图 9 所示。可见在开启两级透明大页的情况下,优化后的虚拟机的分值相较于优化前提升了 45.1%,超过了虚拟机手动关闭一级大页的分值,也超过了宿主机使用小页时的分值。

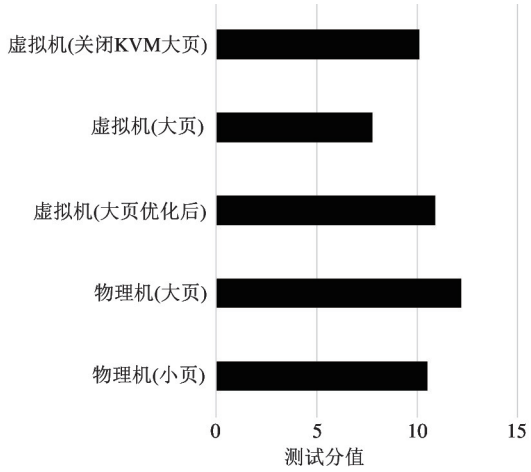

图 9　优化后的 mcf 测试分值

在此基础上,对 SPEC 2006 整体地重新测试,测试结果如图 10 所示。除 mcf 外 astar、cactusADM 和 GemsFDTD 有 1% ~2% 的性能提升,其他测试项变化不明显。没有出现性能下降超过 1% 的项,表明新分配机制引入的开销没有对其他测试的性能产生明显影响。

为了进一步验证优化效果,使用 sysbench 对优化前后的数据库性能进行了对比。sysbench是一个

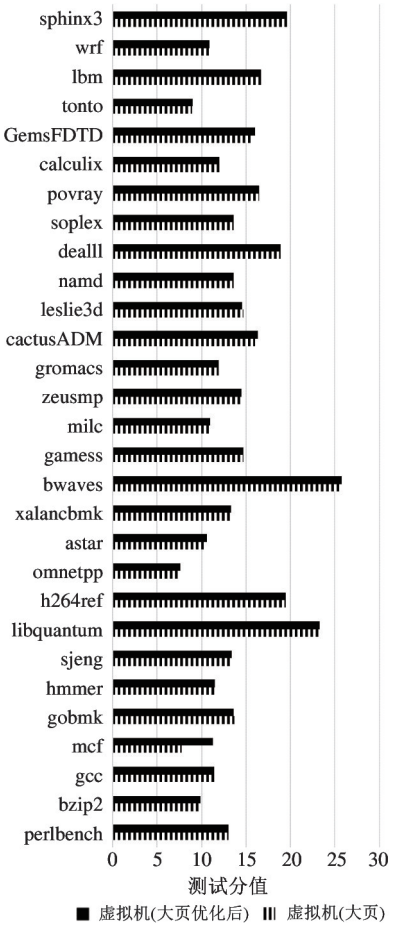

图 10 龙芯 3A4000 优化前后的 SPEC CPU 2006 性能对比

模块化跨平台多线程的基准测试工具,可以评估不同系统参数下的数据库负载情况。对比测试的数据库和测试均在本地运行,测试数据库含 800 万个条目,测试使用 2 个线程,最大数据库事务次数 50 000次,测试时两级透明大页均开启。测试结果如表 1 所示,优化后的测试成绩有 12% 左右的性能提升。

表 1 sysbench 数据库测试成绩对比

	优化前/(次/s)	优化后/(次/s)
数据库事务	153.41	171.74
读写	2914.78	3263.49
其他操作	306.82	343.65

6 结 论

本文利用龙芯 3A4000 处理器和 SPEC CPU 2006 性能测试工具分析了 MIPS VZ 虚拟化环境下大页的性能,并针对发现的分组 TLB 负载不均衡的问题提出了改进方法,经测试改进后的透明大页性能在部分场景中有了明显提高。

本文提到的方法还有进一步改进的空间。本文使用的随机拆分大页的方式可能带来性能抖动,未来通过进一步改良 TLB 负载监测机制可以获得更细粒度的信息,实现对热点大页的识别和有针对性的拆分。同时运行多虚拟机时,不仅存在处理器核 TLB 负载不均衡的问题,还存在着不同虚拟机之间大页分配不均衡的问题;系统长时间运行后,由于不可移动的页面存在会使物理地址空间不连续难以分出大页[13]。这些问题有待后续研究。

参考文献

[1] 胡伟武, 汪文详, 吴瑞阳, 等. 计算机体系结构[M]. (第 2 版). 北京:清华大学出版社, 2017: 215-217

[2] 亨尼西,帕特森,著. 计算机体系结构:量化研究方法[M]. (第 5 版). 贾洪峰,译. 北京:机械工业出版社, 2013: 79-81

[3] Bhargava R, Serebrin B, Spadini F, et al. Accelerating two-dimensional page walks for virtualized systems[J]. *ACM SIGPLAN Notices*, 2008,43(3): 26-35

[4] 王俊儒. 基于龙芯 3A3000 的虚拟机访存优化技术研究[D]. 北京:中国科学院大学, 2019: 15-16

[5] 吴瑞阳, 汪文祥, 王焕东, 等. 龙芯 GS464E 处理器核架构设计[J]. 中国科学 · 信息科学, 2015,45(4): 480-500

[6] Bugnion E, Nieh J, Tsafrir D. Hardware and software support for virtualization[J]. *Synthesis Lectures on Computer Architecture*, 2017, 12(1): 206

[7] Ahn J, Jin S, Huh J. Revisiting hardware-assisted page walks for virtualized systems[J]. *ACM SIGARCH Computer Architecture News*, 2012, 40(3):476-487

[8] 陈华才. 用“芯”探核:基于龙芯的 Linux 内核探索解析[M]. 北京:人民邮电出版社, 2020: 168-169

[9] Papadopoulou M M, Tong X, Seznec A, et al. Prediction-based superpage-friendly TLB designs[C]// 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), Burling, USA, 2015: 210-222

[10] Cox G, Bhattacharjee A. Efficient address translation for

architectures with multiple page sizes[J]. *ACM SIGARCH Computer Architecture News*, 2017, 51(1):435-448

[11] Panwar A, Bansal S, Gopinath K. HawkEye: efficient fine-grained OS support for huge pages[C]// The 24th International Conference on Architectural Support for Programming Languages and Operating Systems, Providence, USA, 2019: 347-360

[12] Kwon Y, Yu H, Peter S, et al. Coordinated and efficient huge page management with ingens[C]// Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, Savannah, USA, 2016:705-721

[13] Panwar A, Prasad A, Gopinath K. Making huge pages actually useful[C]// Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems, Williamsburg, USA, 2018:679-692

Performance optimization of transparent huge pages in MIPS VZ virtualization environment

Zhu Chen[*,**,***,****], Wang Jian[*,**,***], Gao Xiang[****], Mao Bibo[****], Yang Xiaojuan[****]

([*]State Key Laboratory of Computer Architecture, Institute of Computer Technology, Chinese Academy of Sciences, Beijing 100190)

([**]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([***]University of Chinese Academy of Sciences, Beijing 100049)

([****]Loongson Technology Corporation Limited, Beijing 100190)

Abstract

Memory address translation has always been one of the important factors affecting computer performance. This problem is even more prominent due to multiple address translations in a virtualized environment. Using transparent huge pages (THP) is a common method to optimize this problem. This article uses the Loongson 3A4000 processor and its KVM virtual machine as an experimental platform to analyze the performance of the virtual machine using transparent huge pages in the MIPS VZ environment. It is found that transparent huge pages have unbalanced packet TLB load in some applications, which has a negative impact on performance in some scenarios. This article modifies the KVM memory management. By increasing the consideration of packet TLB load in memory management, the performance of KVM virtual machines using transparent huge pages is improved. This method also provides reference for other processors that use different TLBs packet structure.

Key words: transparent huge page (THP), virtualization, MIPS, memory management