ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.12, December 2010, pp.3186−3198
doi: 10.3724/SP.J.1001.2010.03715

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

虚拟机环境下软件按需部署中的预取机制*

陈 彬[1,2+], 肖 侬[1], 蔡志平[1], 王志英[1]

[1](国防科学技术大学 计算机学院,湖南 长沙 410073)

[2](北京军区司令部,北京 100041)

Prefetch Mechanism for On-Demand Software Deployment in Virtual Machine Environments

CHEN Bin[1,2+], XIAO Nong[1], CAI Zhi-Ping[1], WANG Zhi-Ying[1]

[1](School of Computer, National University of Defense Technology, Changsha 410073, China)

[2](The Headquarters of the Beijing Military Area Command, Beijing 100041, China)

+ Corresponding author: E-mail: chenbin@nudt.edu.cn

Chen B, Xiao N, Cai ZP, Wang ZY. Prefetch mechanism for on-demand software deployment in virtual machine environments. *Journal of Software*, 2010,21(12):3186−3198. http://www.jos.org.cn/1000-9825/3715.htm

Abstract: To achieve on-demand software deployment in large-scale virtual machine (VM) environments, this paper presents a prefetch-based on-demand software deployment optimization mechanism to reduce the VM startup latency and to improve the VM running performance at user side. Based on the characteristics of user's behavior of using software and fine-grained splitting of virtual disk (VD) image, the paper prefetches those small-sized being-used VD images at user side from server side by using an access frequency and priority-based prefetch target recognition algorithm—AFPTR and uses a dynamic adjustment mechanism to adjust the prefetch amount during the prefetching process so as to improve the local hit ratio of virtual disk accessing and then the VM running performance at user side. Based on QEMU virtual machine monitor and Linux system, a prototype is built to implement the prefetch mechanism. Experiments on the prototype show that the prefetch mechanism can effectively reduce the VM startup latency and improve the VM running performance at user side, supporting on-demand, fast software deployment in virtual machine environments.

Key words: virtual machine; virtual machine monitor; COW(copy-on-write) disk; prefetch; on-demand software deployment

摘 要: 针对大规模虚拟机环境下软件的按需部署,提出了一种基于预取的按需软件部署优化机制,能够降低用户端虚拟机的启动延迟以及为用户提供更好的虚拟机本地运行性能.基于用户使用软件的行为特点以及虚拟磁盘映像的细粒度分割,预取机制在后台对服务器端存储的虚拟磁盘映像进行预取,通过一种基于访问频率和优先级的预取目标识别算法 AFPTR(access frequency and priority-based prefetch target recognition)和一种预取量动态调节机

* Supported by the National Natural Science Foundation of China under Grant Nos.60736013, 61025009, 61070198, 60903040 (国家自然科学基金); the National Basic Research Program of China under Grant No.2007CB310900 (国家重点基础研究发展计划(973)); the Program for New Century Excellent Talents in University of China under Grant No.NCET-08-0145 (新世纪优秀人才支持计划)

Received 2009-01-04; Revised 2009-06-01; Accepted 2009-07-23

制,将预取集中在用户使用的少数小尺寸的虚拟磁盘映像上,并在预取过程中对预取量进行动态自适应地调节,以提高虚拟磁盘访问的本地命中率,进而提高用户端虚拟机的运行性能.基于QEMU虚拟机和Linux平台,实现了基于预取的按需软件部署原型系统.实验结果表明,预取机制能够有效地降低虚拟机的启动延迟,并能提高虚拟机的本地运行性能,支持虚拟机环境下按需、快速的软件部署.

关键词: 虚拟机;虚拟机管理器;COW(copy-on-write)磁盘;预取;按需软件部署

中图法分类号: TP393 **文献标识码:** A

近年来,系统虚拟机(system virtual machine)[1]因其在解决异构性、可移动性和系统管理等问题方面具有的优势,成为研究和应用的热点,而基于虚拟机实现软件部署已成为快速构建大规模虚拟计算环境的一种较好的解决方案.通过虚拟机管理器VMM(virtual machine monitor),虚拟机的存储设备的整个状态能够被封装到宿主机中的虚拟磁盘映像文件中.虚拟磁盘映像文件可以被传输到其他机器上,然后通过VMM可以快速重建相同的虚拟运行环境,省去了用户重新安装/配置软件的环节.通过VMM和传输易于复制/移动的虚拟磁盘映像文件来替代传统的软件安装/配置过程,能够大大加速软件部署进程,降低软件部署开销,对于大规模计算环境的快速构建、失效恢复等具有重要的意义.

然而,传统的将安装有所有必要软件的单个大尺寸的虚拟磁盘映像分发给每个用户的部署方法存在磁盘空间浪费、传输开销高以及新增软件部署困难等局限性,可以利用基于COW(copy-on-write)机制的块共享技术,在单个软件的基本粒度上将单个大尺寸虚拟磁盘映像分割成多个小尺寸的虚拟磁盘映像(简称COW磁盘)[2],以降低传输开销.在COW模式下,最初生成的COW磁盘称为根COW磁盘,通常用于安装操作系统和最常用的软件.然后,基于根COW磁盘先后生成多个子/孙COW磁盘,每个COW磁盘安装有一种或少数几种软件以使得尺寸尽可能地小.COW磁盘之间的树型层次结构对虚拟机中的客户操作系统(guest OS)是透明的,客户操作系统只能看到完整的一块磁盘.通过按需地向用户分发较小尺寸的COW磁盘,可以大幅降低软件部署的开销,并且COW磁盘中安装的软件在用户端具有较好的本地运行性能.

用户等待虚拟机完成启动、所需软件处于可用状态的时间可称为虚拟机的启动延迟.对于按需分发COW磁盘的部署方法,虚拟机的启动延迟包含COW磁盘传输、解压以及系统启动 3 个过程,用户使用软件之前必须等待这 3 个过程结束.虚拟机的启动延迟会随着为用户部署软件的种类或数量的不同而不同.当用户请求了较多的COW磁盘,或者在网络带宽较低的虚拟机环境下,或者在用户较多的高峰期,启动延迟将会随之增大.通过完全按需取块[3]的方法可以消除COW磁盘传输导致的等待时间,从而大幅降低虚拟机的启动延迟.但由于对COW磁盘中每个磁盘块的首次访问都需要通过网络进行,COW磁盘中软件的运行性能将会大为降低.

基于前期工作对VMM中COW虚拟块设备的优化和细粒度分割机制的研究[4],本文提出了一种优化的、基于预取的按需软件部署机制以降低用户端虚拟机的启动延迟,且能够为用户提供较好的虚拟机本地运行性能.预取机制利用按需取块方法消除COW磁盘传输导致的等待时间以降低启动延迟,同时在后台对用户所使用软件对应的COW磁盘进行预取以提高COW磁盘访问的本地命中率,从而使得COW磁盘中安装的软件具有较好的本地运行性能.基于QEMU[5]虚拟机和Linux平台,我们实现了基于预取的按需软件部署原型系统,并通过实验验证了预取机制的有效性.

本文第 1 节详细介绍基于预取的按需软件部署机制.第 2 节介绍预取的策略和算法.第 3 节介绍原型系统的实现和实验.第 4 节介绍相关工作.第 5 节总结全文.

1 基于预取的按需软件部署优化

针对大规模虚拟机环境下软件的按需部署,我们设计了预取机制来优化基于 COW 磁盘按需分发的软件部署方法.利用预取机制,用户能够从未缓存任何磁盘块的用户端直接启动虚拟机,不需要等待虚拟磁盘映像从服务器端完全传输到用户端;同时,预取机制也能够保证虚拟机在用户端具有较好的本地运行性能.预取机制是基于用户使用软件的行为特点来实现的:

(1) 目的性:用户在一次虚拟机会话中通常只会使用一种或少数几种软件来完成相应的工作;

(2) 重复性:用户通常会重复使用软件的同一功能,从而使得虚拟机运行软件的同一功能模块;

(3) 间断性:用户和软件的交互是一个时断时续的过程,交互触发了虚拟机对虚拟磁盘的访问.

用户是指使用虚拟机中软件的对象,可以是真实用户,也可以是一种软件或服务.根据目的性特点,如果仅仅将用户使用的少数几种软件对应的磁盘块传输到用户端,那么将大幅降低部署的开销,且能够保证软件运行过程中的大多数磁盘访问在用户端本地进行,因而具有较好的运行性能.根据重复性和间断性特点可知,在虚拟机运行过程中存在一些(甚至大量)时间区段,虚拟机不会访问虚拟磁盘;对于真实用户,用户和软件的两次交互之间的时间间隔通常在秒的量级.因此,可以利用这些空闲的时间区段在后台预取更多的磁盘块到用户端,使得尽可能多的磁盘访问在本地进行,从而提高虚拟机中软件的运行性能.

预取机制主要通过按需取块过程和预取过程相结合的方式实现的.在虚拟机运行过程中,预取机制通过按需取块过程按需地从服务器端获取虚拟机请求的但未在本地缓存的磁盘块,通过预取过程在后台预取更多的磁盘块到用户端,以使得尽可能多的磁盘访问在本地进行.一旦用户使用的软件对应的所有磁盘块通过按需取块和预取过程完全缓存到本地,则与将虚拟磁盘完全传输到本地一样,用户能够获得较好的软件运行性能.

预取机制包含的主要组件如图 1 所示.虚拟机管理器 VMM 和预取客户程序 cow_client 运行在用户端.服务器端运行预取服务程序 cow_server 以及存储所有的 COW 磁盘.预取客户程序 cow_client 是实现预取机制的关键,它主要包含 3 个模块:按需取块模块(demand_fetch)、预取模块(prefetch)以及预取策略模块(prefetch_policy),分别实现按需取块过程、预取过程以及特定的预取策略.预取服务程序 cow_server 接收来自 cow_client 的取块请求,从服务器端存储的 COW 磁盘中读取相应的磁盘块,并发送给用户端的 cow_client;cow_client 将接收到的块写入本地块缓存(block cache)或直接交给 VMM.本节将详细介绍基于预取的按需软件部署优化机制.

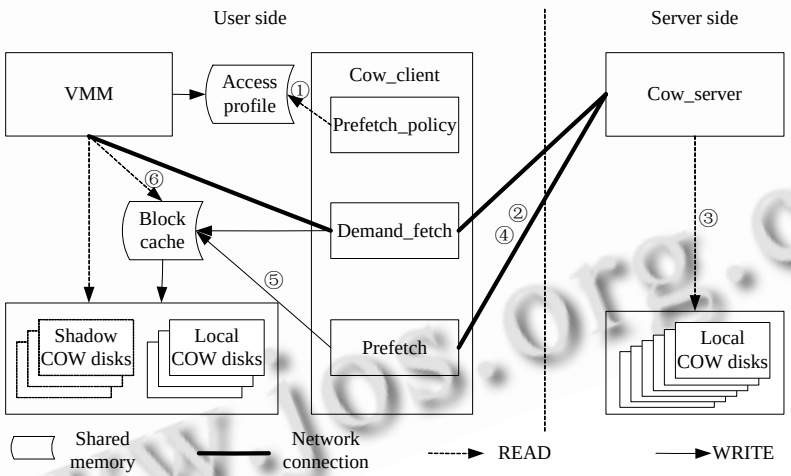

Fig.1 Main components in prefetch-based on-demand software deployment

图 1 基于预取的按需软件部署中的主要组件

1.1 VMM中COW虚拟块设备的优化和细粒度分割

如前所述,用户在一次虚拟机会话中通常只会使用一种或少数几种软件,因此在块设备层通常也只有整个虚拟磁盘的一小部分磁盘块会被访问到.如果预取过程能够实时且仅仅预取用户所使用软件对应的这一小部分磁盘块,则预取的针对性将会大大提高,有助于减少无效预取.对于传统的单个大尺寸虚拟磁盘映像,用户所使用软件的不确定性以及VMM和客户操作系统之间的语义鸿沟(semantic gap)[6]使得难以在VMM层实时地将用户所使用的软件和其底层占用的磁盘块进行对应,因此难以实现有针对性地预取.

基于前期工作对VMM中COW虚拟块设备的优化[4],我们能够在不影响虚拟磁盘访问性能的前提下,在单个软件的基本粒度上将传统的单个大尺寸虚拟磁盘映像分割成多个小尺寸的COW磁盘映像.通过这种细粒度

的分割,软件与其对应的磁盘块(COW磁盘)之间的对应关系已经显式地建立.如果能够在预取过程中实时地识别用户所使用软件对应的COW磁盘,即预取目标,并将预取集中到这些少数几个COW磁盘上,预取的针对性将会大大提高.本文第 2.2 节介绍了一种能够实时地识别用户所使用软件对应的COW磁盘的预取目标识别算法.

1.2 按需取块

虚拟机在用户端运行过程中,预取机制利用按需取块方法从服务器端获取暂未在本地缓存的磁盘块.通过按需取块,用户能够直接启动虚拟机,不需要等待COW磁盘从服务器端完全传输到本地,可以大幅降低虚拟机的启动延迟.对于每个未在本地缓存的COW磁盘,会在用户端生成一个影子COW磁盘(shadow cow disk)[2],用于保存从服务器端对应COW磁盘中获取的磁盘块,如图 1 所示.从服务器端按需获取的磁盘块首先被发送给VMM以满足读请求,然后被保存到块缓存(block cache)中,并最终被写回到对应的影子COW磁盘中.

按需取块过程主要由图 1 中的 cow_client 的 demand_fetch 模块和 cow_server 程序来实现,是一个 C/S 模式的交互过程.设 request_blocks 函数在发生缺块时发送取块请求,read_blocks 函数从服务器端存储的 COW 磁盘中读取用户端请求的磁盘块,send_blocks 函数发送读取的磁盘块,save_blocks 将接收到的磁盘块保存到本地,则按需取块过程中 VMM,demand_fetch 以及 cow_server 之间的交互过程如图 2 所示.

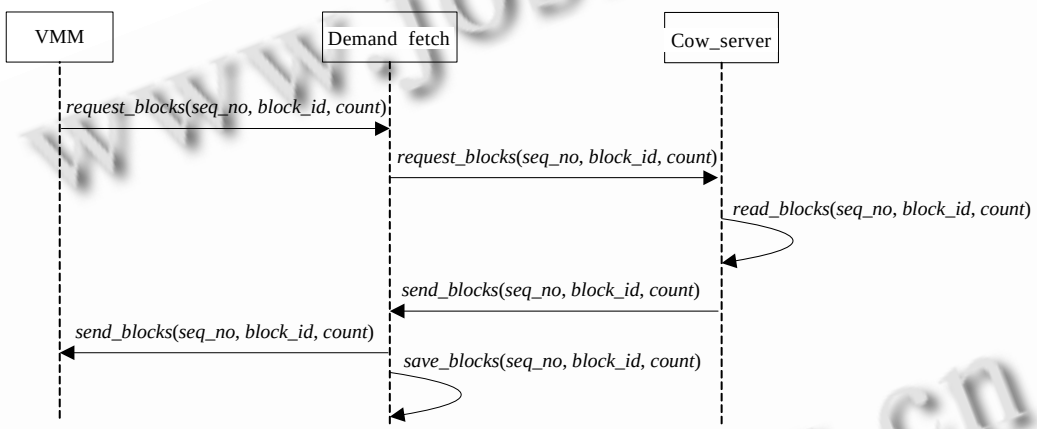

Fig.2 On-Demand block fetching process

图 2 按需取块过程

设软件A的COW磁盘为COW_a,在用户使用A的过程中,VMM接收到L次对COW_a的不同读请求,其中,M次读请求是VMM通过读取本地COW_a的影子COW磁盘或块缓存中的块来满足的,N次读请求是VMM通过按需取块过程从服务器端获取全部或部分请求的块来满足的,则我们定义访问COW_a的本地命中率$LHR=M/L$,且它们满足关系:(1) $L=M+N$; (2) $LHR=M/L=1-N/L$.

由图 2 可知,按需取块是一个同步过程,在请求的磁盘块未传输到用户端之前,VMM 和虚拟机必须等待.在相同的运行条件下,按需取块的次数就越多,COW 磁盘访问的本地命中率越低,COW 磁盘中软件的运行性能就越低.我们通过第 1.3 节的预取过程来减少按需取块的次数,尽可能提高 COW 磁盘访问的本地命中率.

1.3 预 取

与按需取块过程不同,预取过程在后台异步地进行,对 VMM 是透明的.预取过程的优先级比按需取块过程低,它只在无按需取块时进行.一旦 VMM 有按需取块请求,预取过程将暂停.预取过程由图 1 中的 cow_client 的 prefetch 模块和 cow_server 程序实现,主要包含如下步骤:

(1) 预取策略模块 prefetch_policy 执行特定的预取策略,以确定预取目标(COW 磁盘)、预取位置(起始块号)以及预取量;

(2) 当 VMM 无按需取块请求时,预取模块 prefetch 根据 prefetch_policy 模块执行的结果,向服务器端的

预取服务程序 cow_server 发送取块请求;

(3) 服务器端的 cow_server 接收到请求后,从相应的 COW 磁盘中读取块并发送给用户端的 prefetch 模块;然后返回继续等待取块请求;

(4) 用户端的 prefetch 模块将接收到的块写入块缓存(block cache),然后继续下一轮预取,即执行步骤 1~步骤 4.

从预取过程可以看到,预取模块 prefetch 没有和 VMM 直接交互,而是将预取的块写入块缓存后继续预取,不需要 VMM 等待.VMM 读取块时,首先查询本地的块缓存;如果缓存未命中,则从 COW 磁盘中直接读取.

预取策略模块 prefetch_policy 用于实现特定的预取策略和算法,以确定预取目标、预取位置以及预取量.确定预取目标实际上就是确定预取的 COW 磁盘,这就需要在 VMM 层实时和准确地识别用户所使用软件对应的 COW 磁盘.确定了预取的目标 COW 磁盘之后,还需要确定从目标 COW 磁盘中何处开始预取.面对变化的网络环境和不同种类的软件,选择合适的预取量对于提高预取的效果具有重要的意义.第 2 节详细介绍了我们在预取策略模块 prefetch_policy 中实现的预取策略和算法.

用户端的块缓存(block cache)是由 VMM 和预取客户程序 cow_client 共享的一块内存区域,用于提高 VMM 读写虚拟磁盘的性能.按需取块模块 demand_fetch 和预取模块 prefetch 都会将从服务器端获取的块保存到块缓存中.当缓存的块超过一定数量时,VMM 采用 LRU(least recently used)替换策略将部分磁盘块写回对应的影子 COW 磁盘.VMM 和预取客户程序 cow_client 通过锁机制来实现对块缓存的读写互斥.

2 预取策略和算法

2.1 预取策略

好的预取策略能够有效减少按需取块的次数,提高虚拟磁盘访问的本地命中率.预取策略主要包括如下 3 个方面:

(1) 预取目标:根据目的性特点,我们将用户所使用软件对应的少数几个 COW 磁盘作为预取目标,并设计了一种基于访问频率和优先级的预取目标识别算法 AFPTR(access frequency and priority-based prefetch target recognition)来识别预取的目标 COW 磁盘.

(2) 预取位置:基于访问局部性原理,我们将预取目标 COW 磁盘中与上次被访问或预取的块最相邻的且未在用户端缓存的块作为下次预取的起始块.

(3) 预取量:由于预取受网络条件、软件种类、VMM 最大块请求量等诸多因素影响,且有些因素是不断变化的,因此预取过程有必要根据这些因素的变化对预取量进行动态调节,力求取得更好的预取效果.我们设计了一种动态调节机制,能够根据网络条件和软件读请求的变化对预取量进行动态自适应地调节,以降低各种因素对预取的影响,进而提高预取效果.

由于用户可能一次使用多种软件,因此预取目标也需要随着用户使用软件的变化而变化.基于 VMM 的磁盘访问统计信息,prefetch_policy 模块通过 AFPTR 算法(详见第 2.2 节)能够较为准确地识别用户当前所使用软件对应的 COW 磁盘,从而使得预取目标能够随着用户使用软件的变化而变化.prefetch_policy 模块利用 VMM 的磁盘访问统计信息以及跟踪按需取块和预取过程来确定每次预取的起始块号.确定预取量时,prefetch_policy 模块利用 VMM 的磁盘访问统计信息获得外部网络条件和内部软件读请求的变化,然后对预取量进行动态地调节(详见第 2.3 节).由于网络带宽的限制以及 COW 磁盘中的磁盘块可能不连续,预取量并非越大越好,较大预取量导致的较高传输或分片开销反而可能导致本地命中率下降.

2.2 基于访问频率和优先级的预取目标识别算法——AFPTR

预取目标的准确性对于提高预取的针对性、进而提高 COW 磁盘访问的本地命中率具有重要意义.确保预取目标准确性的关键是如何在 VMM 层实时和准确地识别用户所使用软件对应的 COW 磁盘.如果一个 COW 磁盘在较短时间内的访问频率高于其他 COW 磁盘,那么其中安装的软件很可能就是用户正在使用的软件,该

COW 磁盘应该作为当前的预取目标;如果一个 COW 磁盘在一段较长的时间内访问频率较高,那么其中安装的软件很可能就是用户正在或频繁使用的软件,其成为预取目标的优先级应该较高.基于上述思想,我们设计了一种基于访问频率和优先级的预取目标识别算法 AFPTR.AFPTR 算法通过对 VMM 层 COW 磁盘访问的统计信息进行分析,将访问频率最高或优先级最高的 COW 磁盘作为预取目标.AFPTR 算法的基本思想如下:

(1) 时间分片:将用户使用软件的时间分成长度为 *t* 的较短时间片 TS(time slice),单个时间片内的磁盘访问统计信息是 AFPTR 识别预取目标的基本依据.

(2) 当前预取目标:AFPTR 根据上一个时间片的 VMM 磁盘访问统计信息,将访问频率最高的 COW 磁盘作为当前时间片的预取目标;当用户使用多种软件时,当前预取目标会随着使用软件的变化而变化.

(3) COW 磁盘优先级:每个 COW 磁盘被赋予一个优先级;当一个 COW 磁盘在一个时间片内由于访问频率最高而成为预取目标时,递增该 COW 磁盘的优先级;当一个 COW 磁盘在 *N* 个连续的时间片内未成为预取目标时,递减该 COW 磁盘的优先级.

(4) 超时机制:对于无读请求的时间片,将上个时间片的预取目标作为当前预取目标;在连续经过 *M* 个无读请求的时间片后,如果仍然无读请求,则将优先级最高的 COW 磁盘作为当前预取目标.

VMM 将每个时间片的 COW 磁盘访问统计信息(如被访问次数等)记录在如图 1 所示的共享内存区 access_profile 中.当一个 COW 磁盘被访问时,VMM 递增 access_profile 中相应的计数值.利用 access_profile,预取策略模块 prefetch_policy 每个时间片执行一次 AFPTR 算法以确定当前时间片的预取目标 COW 磁盘.预取模块 prefetch 根据预取目标 COW 磁盘,在 VMM 无按需取块请求的情况下执行预取过程.

设 *pre_target* 和 *cur_target* 分别为上一个时间片和当前时间片的预取目标 COW 磁盘的序列号(COW 磁盘生成顺序编号,用于标识 COW 磁盘);*access_profile*,*TS_counter* 和 *cow_prio* 都是以 COW 磁盘序列号为下标的数组;*access_profile* 记录每个 COW 磁盘的被访问次数;*TS_counter* 记录每个 COW 磁盘的时间片计数值;*cow_prio* 记录每个 COW 磁盘的优先级;函数 *Inc_Slice_Counter* 递增指定 COW 磁盘的时间片计数,当计数值为 *N* 时,清零时间片计数并递减其优先级;函数 *Get*_Max 返回给定数组中的最大值;函数 *Fetch_Complete* 判断给定的 COW 磁盘是否已完全传输到本地;函数 *Clear_Access_Profile* 清除 *access_profile* 中上一个时间片的 COW 磁盘访问计数;*M* 和 *N* 为超时阈值;*m* 为计数值,初值为 0.AFPTR 算法的伪代码描述见算法 1.

算法 1.

输入:COW 磁盘访问次数数组 *access_profile*,COW 磁盘时间片计数数组 *TS_counter*,COW 磁盘优先级数组 *cow_prio*,超时阈值 *M*,*N*.

输出:预取目标 COW 磁盘的序列号 *cur_target*.

Procedure *AFPTR*()

```
1.  Inc_Slice_Counter(TS_counter);
2.  cur_target=Get_Max(access_profile);          //将访问频率最高的 COW 磁盘作为预取目标
3.  if (cur_target>0 && !Fetch_Complete(cur_target)) {
4.      cow_prio[cur_target]++;
5.      TS_counter[cur_target]=0;
6.      m=0;
7.  } else {                                     //VMM 无读请求
8.      if (cur_target==0 ) {
9.          cur_target=pre_target;
10.         m++; }
11.     if (m==M||Fetch_Complete(cur_target)) {
12.         cur_target=Get_Max(cow_prio);        //超时,将优先级最高的 COW 磁盘作为预取目标
13.         if (cur_target==0)
```

```
14.                cur_target=1;                    //无其他预取目标,将根 COW 磁盘作为预取目标
15.            m=0;}}
16.  Clear_Access_Profile();
17.  pre_target=cur_target;
18.  return cur_target;
```

AFPTR 算法通过细粒度的时间分片和基于访问频率的预取目标识别方法实时地识别用户当前所使用软件对应的COW磁盘,提高了预取的针对性;通过动态调整COW磁盘的优先级,使得在一段较长的时间内访问频率越高的 COW 磁盘成为预取目标的几率越大,进一步提高了预取目标的准确性;通过设置合适的超时参数实现了利用访问局部性和高优先级 COW 磁盘优先预取之间的平衡,尽可能减少针对错误预取目标的无效预取.相比较于传统的单个大尺寸虚拟磁盘映像,AFPTR 算法通过将预取集中在用户所使用软件对应的少数几个COW磁盘上,极大地提高了预取的针对性,有助于减少按需取块的次数,提高虚拟机在用户端的运行性能.

2.3 预取量动态调节机制

对于预取过程中预取量的调节,我们主要考虑了如下两个因素:

(1) 网络条件:网络条件是影响预取的外部因素.网络带宽越宽,延迟越低,预取过程应采用较大的预取量;反之,应采用较小的预取量.

(2) 软件读请求特点:软件运行过程中的读请求特点是影响预取的内部因素.对于读请求密集的时段,较大的预取量能更有效地提高本地命中率;反之,应适度减小预取量以降低预取对虚拟机性能的影响.

我们用网络传输速率的变化比例来衡量网络条件的变化程度.网络传输速率NS(network speed)可以通过预取过程测量得到.若一次预取n个块所花时间为T,则$NS=n/T$.设前后两个TS(时间片)中测量的NS分别为NS_0和为NS_1,则网络传输速率的变化比例ΔNS可根据公式(1)来确定:

$$\Delta NS=(NS_1-NS_0)/NS_i,若NS_1<NS_0,NS_i=NS_1;否则,NS_i=NS_0 \tag{1}$$

我们用本地缓存失效次数的变化比例来衡量软件读请求的变化程度.本地缓存失效次数MC(missing count)可以通过测量每个TS内按需取块的次数获得.设前后两个TS中按需取块的次数分别为MC_0和MC_1,则

$$\Delta MC=(MC_0-MC_1)/MC_i,若MC_1<MC_0,MC_i=MC_1;否则MC_i=MC_0;若MC_i=0,MC_i=1 \tag{2}$$

为了使得一次预取能够尽可能至少满足一次读请求,我们将VMM最大块请求量(Max_*Req_Count*)作为预取量的最低限.对于特定的VMM,其最大块请求量是常量,可以在虚拟机运行过程中通过测量得到.在每个TS中,根据NC和MC的增量以Max_*Req_Count*为单位对预取量进行调整.根据我们的实验结果,当预取量大于一定值时,本地命中率会明显下降.为了避免由于NS和MC的剧烈变化造成预取量的过大或过小,我们设计了有界的随指数变化的增量调节因子η来控制预取量的变化.设T为平均增量,如公式(3)所示,则预取量增量调节因子η根据公式(4)来确定.设预取量初值为 2Max_*Req_Count*,上一次的预取量为P_{n-1},则当前TS的预取量P_n根据公式(5)来确定:

$$T=(\Delta NC+\Delta MC)/2 \tag{3}$$

$$\eta=(-1)^t\times[10-20e^{T/8}/(e^{T/4}+1)],若T>0,则t=0;否则t=1 \tag{4}$$

$$P_n=\eta\times\text{Max\_}Req\_Count+P_{n-1},若P_n<\text{Max\_}Req\_Count,则P_n=\text{Max\_}Req\_Count \tag{5}$$

3 实现与实验

3.1 实 现

基于 QEMU 虚拟机和 Linux 平台,我们实现了基于预取机制的按需软件部署原型系统,并通过实验验证了预取机制的有效性.

我们对前期工作[4]在QEMU中开发的COW虚拟块设备驱动mcow进行了扩展以支持预取.在mcow中实现了和外部通信的接口以支持按需取块,增加了对共享内存区的管理功能:对用户端的块缓存(block cache)采用

LRU替换策略进行管理以及实时更新共享内存区(access_profile)中QEMU的磁盘访问统计信息.

我们开发了预取客户程序 *cow_client* 和预取服务程序 *cow_server* 来实现预取机制.*cow_server* 是一个运行在服务器端的采用多线程实现的服务进程,接收从用户端 *cow_client* 发来的取块请求(按需取块和预取),从服务器端指定的COW磁盘中读取指定的块,然后返回給 *cow_client*.*cow_client* 运行在用户端宿主操作系统(host OS)中,主要由预取策略模块 *prefetch_policy*、按需取块线程 *demand_fetch* 以及预取线程 *prefetch* 这 3 部分来实现.

3.2 实 验

实验中的服务器和用户机均配置有主频为 1.8GHZ 的 Intel 双核处理器和 1GB 内存,并通过 100Mbps 的以太网相连.宿主和客户操作系统均为 Fedora 7.经过扩展的包含 mcow 块设备驱动的 QEMU 0.9.0 作为 VMM 部署在服务器端和用户端.虚拟机配置了 512MB 内存和 10GB 的虚拟磁盘(根分区 9GB,文件系统 ext3,块大小 4KB).我们准备了 3 种常用软件的 rpm 包用于生成 COW 磁盘,并将生成的 COW 磁盘存储在服务器端.在实验过程中,我们利用安装在根 COW 磁盘中的宏记录/回放工具 JW_Record_Playback 来自动运行预先定义的软件常用功能.表 1 列出了实验中使用的软件及其常用功能的定义.

Table 1 COW disks and definition of commonly-used software functions

表 1 COW 磁盘及软件常用功能定义

seq_no	COW disk	Size (compressed)	Commonly-Used functions
1	F7	688.3 MB	OS booting
2	Gimp 2.2	29.5 MB	Regular functions*, Layer(New,Copy,Delete,Merge), Filter(Blur,Enhance, Twist,General,Noise), Select Tools, Painting Tools, Transform Tools
3	Openoffice.writer	79.2 MB	Regular functions*, Export, Find & Replace, Table, Drawing, Insert(Header,Footer,Special Character,Picture), Character, Column, Paragraph, Bullets and Numbering, Page, Options
4	Adobe reader 7.01	52.1 MB	Regular functions*, Save as Text, Properties, Select All, Find, Search, Preferences, Page Display, Basic Tools, Zoom Tools

* Regular functions: Open, Close, Save, Copy, Cut, Paste, Save as

3.2.1 根 COW 磁盘部署

根COW磁盘由于安装有操作系统,需要首先部署到用户端.由于尺寸较大,根COW磁盘通常会导致较高的部署开销.针对根 COW 磁盘的部署,我们比较了在如下两种部署方法下虚拟机的启动延迟:

(1) 传统方法:将根 COW 磁盘从服务器端传输到用户端,解压缩,然后启动虚拟机.

(2) 预取方法:用户端不缓存任何磁盘块,通过按需取块过程和预取过程相结合的方式启动虚拟机.

对每种方法,我们测试 3 次并取平均值作为最后的结果.实验结果见表 2.从结果可以看到,预取方法降低了约 81.4%的启动延迟.根据测量结果,通过预取方法启动虚拟机后,用户端根 COW 磁盘对应的影子 COW 磁盘的实际大小约为 198.4MB,是整个根 COW 磁盘大小的 7.9%.而且,按需取块过程在系统启动过程中占有较大的比例,预取过程占有较小的比例,大约只有 5.2%的磁盘块是通过预取获得的.我们认为,这主要是因为系统启动过程中磁盘访问频繁且预取过程具有较低的优先级.按需取块过程由于避免了传输大量不必要的磁盘块,因而能够节省大量的时间开销,降低虚拟机的启动延迟.对于其他 COW 磁盘的部署,在传统方法下,更多 COW 磁盘的传输和解压缩带来的开销会进一步增加启动延迟;而在预取方法下,由于虚拟机的启动延迟只包括系统启动的时间开销,没有 COW 磁盘传输和解压缩的时间开销,因此不会增加启动延迟.

Table 2 VM startup latency (s)

表 2 虚拟机启动延迟 (秒)

	Transfer time	Decompress time	Boot time	Startup latency	Decreased amplitude
Traditional	89.2	382.4	82.3	553.9	81.4%
Prefetch	0	0	102.5	102.5	

3.2.2 预取目标识别

通过预取机制将根 COW 磁盘部署到用户端之后,我们对表 1 中 3 种软件的 COW 磁盘分别测量了在如下

3 种部署方法下 COW 磁盘访问的本地命中率:

(1) 完全按需取块(Demand):只有按需取块过程,无预取过程.

(2) 预取,无预取目标识别(No-AFPTR):按需取块和预取过程相结合,并将上次访问的 COW 磁盘作为当前预取目标.

(3) 预取,预取目标识别(AFPTR):按需取块和预取过程相结合,并利用 AFPTR 算法来识别预取目标.

对于每种部署方法,我们运行表 1 中为每种软件定义的常用功能,并测量在该部署方法下的本地命中率.首先,单独运行每种软件以进行单个预取目标的测试,然后轮流运行 3 种软件以进行多个预取目标的综合测试.在方法 1 中,我们屏蔽了预取模块和预取策略模块,只保留了按需取块模块;在方法 3 的 AFPTR 算法中,我们取 t=5s,N=10,M=3;为了比较的公平性,方法 2 和方法 3 中的预取量均取 32KB 的相同预取量.我们在 QEMU 的 mcow 块设备驱动中添加了测试代码以记录每次测试中本地访问和按需取块的次数,并根据第 1.2 节的方法计算本地命中率.对于每次本地命中率的测量,我们测量 3 次并取平均值作为最后的结果.图 3 给出了 3 种部署方法下本地命中率的实验结果对比.从图 3 可以看到,在完全按需取块的情况下,COW 磁盘访问的本地命中率最低,大部分访问需要通过网络进行;在无 AFPTR 的情况下(No-AFPTR),预取方法也能够大幅提高 COW 磁盘访问的本地命中率,平均提高了 20%;在利用 AFPTR 算法识别预取目标的情况下,COW 磁盘访问的本地命中率最高,平均为 83%,最高为 88%,相比较于完全按需取块和无 AFPTR 的情况,分别平均提高了 41.8%和 21.8%.

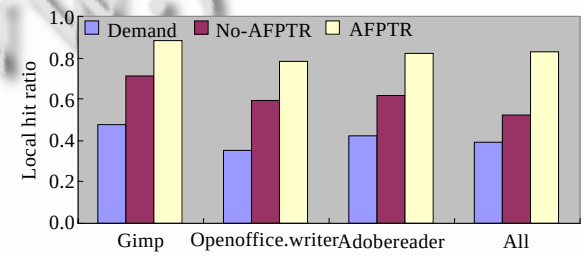

Fig.3 Comparison of local-hit-ratios under different deployment approaches

图 3 不同部署方法下的本地命中率比较

在测量本地命中率的过程中,我们也记录了在方法 2 和方法 3 中每次确定的预取目标.记录的预取目标的百分比统计结果见表 3.从统计结果可以看到,AFPTR 算法确定预取目标的准确性明显高于无 AFPTR 的情况,如表 3 中阴影部分所示.在对 3 种软件的单独测试中,AFPTR 算法识别预取目标的准确性最高为 97%,平均为 85.5%;在综合测试中,只有 3.1%的预取目标被 AFPTR 算法识别为根 COW 磁盘.一种特殊的情况是 Gimp,从统计结果可以看到,只有 64.2%的预取目标被识别为 Gimp 对应的 COW 磁盘,35.1%的预取目标被识别为根 COW 磁盘且集中在测试过程的后期.经过分析我们发现,在测试过程中,Gimp 对应的 COW 磁盘已经被完全预取到用户端,因此在测试过程的后期,AFPTR 算法在没有其他预取目标的情况下,将根 COW 磁盘作为预取目标.

Table 3 Statistical results of prefetch target determination (%)

表 3 预取目标确定的统计结果 (%)

seq_no	COW disk	PT determination method	1	2	3	4
2	Gimp 2.2	AFPTR	35.1	64.2	0.5	0.2
		No-AFPTR	48.5	49	1.6	0.9
3	Openoffice.writer	AFPTR	4.2	0.1	95.4	0.3
		No-AFPTR	9.1	0.6	89.2	1.1
4	Adobe reader 7.01	AFPTR	2.6	0.2	0.2	97
		No-AFPTR	7.2	0.4	0.5	91.9
2, 3, 4	All	AFPTR	3.1	39.7	31.5	25.7
		No-AFPTR	5.8	41.2	30.1	22.9

3.2.3 预取量

为了测试第 2.3 节中预取量动态调节机制的有效性,我们对预取量恒定和预取量动态调节两种策略进行了

对比.在第 3.2.2 节的实验中,我们测量得到 QEMU 的最大块请求量 Max_*Req_Count* 为 2.对于预取量恒定的策略,分别取 1,2,4,6,8 和 10 倍 Max_*Req_Count* 的预取量.运行表 1 中为每种软件定义的常用功能,并测量两种策略下对应 COW 磁盘访问的本地命中率.我们测量了 3 种软件单独运行和轮流运行 4 种情况下的本地命中率.每次测量重复 3 次,并取平均值作为最后的结果.图 4 给出了测量结果,可以看到,在预取量动态调节的策略下(dynamic),4 种情况下的测试都获得了较高的本地命中率,最高为 92%,平均为 85%,因而预取量的动态调节是有效的;在预取量恒定的策略下,4 种情况下的本地命中率随预取量的不同而变化较大,并且在预取量大于一定值时,本地命中率出现了不同程度的下降;不同情况下取得最大本地命中率时的预取量也不尽相同,因此难以在虚拟机运行过程中针对不同种类的软件确定合适的预取量.

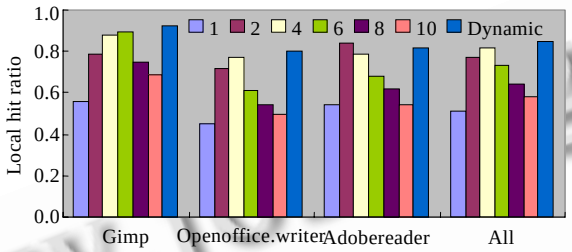

Fig.4 Comparison of local-hit-ratios under different prefetch amount determination policies

图 4 不同预取量确定策略下的本地命中率比较

3.2.4 多用户环境

我们测试了在多用户环境下预取机制的可伸缩性.利用局域网内的 5 台物理机器(不包括服务器和测试用户端)以及 QEMU 来模拟 1~10 个用户的环境,即每台机器最多运行 2 个 QEMU 虚拟机.对于不同的用户数量,每个用户(虚拟机)和测试用户端同时启动并轮流运行表 1 中 3 种软件的常用功能,我们测量用户端虚拟机启动延迟和 COW 磁盘访问本地命中率.图 5 和图 6 分别给出了 1~10 个用户下的用户端虚拟机启动延迟和 COW 磁盘访问本地命中率的测量结果.从测量结果可以看到,用户数量对于用户端虚拟机启动延迟以及本地命中率的影响较小;当用户数为 10 时,启动延迟增加了约 6.9%,本地命中率降低了约 4.7%.值得注意的是,当用户数较少时(2~4),启动延迟略有降低且本地命中率略有增加.我们认为,这是由于其他用户访问服务器产生的服务器端缓存提高了对后续请求的响应速度.通过改善和优化服务器端预取服务程序(*cow_server*)的缓存和请求队列设计,将有助于进一步提高预取机制在多用户环境下的可伸缩性.

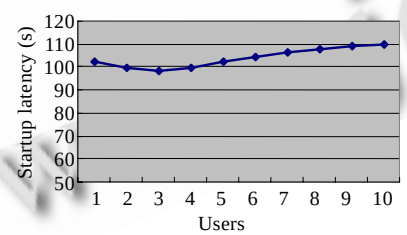

Fig.5 Comparison of VM startup latencies under different number of users

图 5 不同用户数目下的虚拟机启动延迟比较

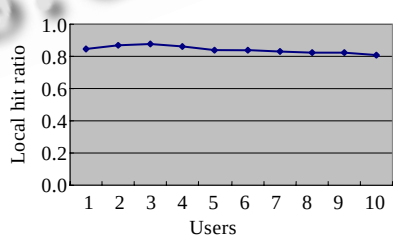

Fig.6 Comparison of local-hit-ratios under different number of users

图 6 不同用户数目下的本地命中率比较

3.2.5 实验小结

通过在原型系统上的实验,我们可以得出下面的结论:

(1) 按需取块方法能够避免传输大量不必要的磁盘块,从而大幅降低虚拟机的启动延迟.

(2) AFPTR 预取目标识别算法能够较准确地识别用户所使用软件对应的 COW 磁盘;相比较于按需取块的部署方法,预取方法,特别是基于 AFPTR 算法的预取方法,能够显著提高 COW 磁盘访问的本地命中率,进而提高虚拟机中软件的运行性能.

(3) 预取量动态调节机制能够在预取过程中基于网络条件和软件读请求的变化对预取量进行动态自适应地调节,有助于提高用户端 COW 磁盘访问的本地命中率,进而获得较好的预取效果.

(4) 预取机制在多用户环境下具有较好的可伸缩性:随着用户数量的增加,用户端虚拟机启动延迟仅有小幅度缓慢地增加,COW 磁盘访问的本地命中率则有小幅度缓慢地降低.通过改善和优化服务器端预取服务程序的缓存和请求队列设计,将有助于进一步提高预取机制在多用户环境下的可伸缩性.

4 相关工作

Stanford大学的Collective[2,7]项目基于虚拟机提出了虚拟装置(virtual appliance,简称VAP)的概念,试图通过虚拟装置降低计算环境中软件管理的开销.由于每个VAP不仅包含软件而且包含操作系统,VAP是比COW磁盘更加重量级的实体,通过网络部署VAP的开销较大.Sapuntzakis等人[2]提出了有效的技术来优化虚拟机在低速网络链路上的迁移;由于COW磁盘层次不能太深以避免虚拟磁盘访问的高开销,某些COW磁盘就会包含较多种类的软件从而使得尺寸变大,进而导致较高的传输开销.基于优化的COW虚拟块设备及其细粒度的分割机制,我们能够在不影响虚拟磁盘访问性能的前提下,生成大量小尺寸的COW磁盘,并通过预取机制进一步降低软件部署的开销,支持虚拟机环境下按需的、快速的软件部署.

Ventana[8]试图通过文件系统虚拟化的方法提高虚拟磁盘的透明度,使得多个虚拟机能够共享底层虚拟磁盘中的文件对象.与Ventana目标类似,Mirage[9]设计了一种新的虚拟磁盘映像格式MIF来优化虚拟机环境下大量虚拟磁盘映像的存储,并能够透明地对虚拟磁盘映像中的软件执行查找、更新等操作.Ventana和Mirage主要针对的是虚拟机非执行状态下虚拟磁盘映像存储管理优化,没有涉及虚拟机运行过程中的部署优化问题.Parallax[10]是一个块级别的分布式虚拟磁盘映像存储系统,每个客户端运行一个存储虚拟机(storage VM);存储虚拟机通过对后端共享块存储(block store)的访问来为前端其他虚拟机提供存储服务;Parallax利用本地的持久磁盘缓存来降低对网络存储的大量突发访问带来的性能下降,没有实现相关的预取机制.

Carnegie Mellon大学和Intel合作的研究项目ISR[3]利用分布式文件系统和虚拟机技术,使得用户能够从网络上不同的站点访问个人的计算环境.ISR利用了按需取块和基于内容寻址的存储(CAS)等多种优化技术来减少通过网络传输的虚拟机状态数据.GVFS[11]扩展了分布式文件系统NFS以支持网格环境下虚拟机状态的按需传输.ISR和GVF采用的按需取块或按需状态传输方法使得用户端虚拟机的运行性能大幅降低;且它们都是针对单个大尺寸的虚拟磁盘映像,难以通过有针对性的预取以有效提高用户端虚拟机的运行性能.PDS[12]提供了一个虚拟执行环境用来按需地部署和执行asset——一种类似于COW磁盘的软件映像.PDS在操作系统和应用之间插入一个虚拟化器(virtualizer)来有选择性地截获部分Windows系统调用,因此实现较为复杂且和平台相关.

国内近年来在虚拟化领域也开展了大量的研究.国家在 2005 年和 2007 年先后启动了 973 项目“虚拟计算环境聚合与协同机理研究”[13−15]和“计算机系统虚拟化基础理论与方法研究”[16,17].北京航空航天大学的CROWN虚拟计算平台CIVIC[17]集成了多种虚拟机技术,旨在为用户提供独立、隔离的计算环境,为管理人员提供硬件资源和软件资源的集中管理功能.华中科技大学提出了虚拟用户环境的组织模型及数据封装模型[14],且为这些模型提供了自动安装部署、在线迁移等必要的操作机制,可实现用户操作环境随操作环境映像数据一起迁移.清华大学的Desktop2Go[15]方法将应用层虚拟化技术引入到软件服务系统之中,设计并实现了程序的虚拟运行环境;并在网络分发过程加入P2P下载功能,能够有效降低服务器负载和充分利用网络资源.

5 结论及展望

本文提出了一种基于预取的按需软件部署优化机制,以降低用户端虚拟机的启动延迟以及为用户提供较

好的虚拟机本地运行性能.预取机制在后台对用户所使用软件的 COW 磁盘进行预取,以提高 COW 磁盘访问的本地命中率,进而提高虚拟机中软件的运行性能.通过一种基于访问频率和优先级的预取目标识别算法 AFPTR 以提高预取目标识别的准确性,减少无效预取.通过一种预取量动态调节机制在预取过程中对预取量进行动态自适应地调节,降低网络条件等变化因素对预取的影响,进而提高预取效果.基于 QEMU 虚拟机和 Linux 平台,我们实现了基于预取的按需软件部署原型系统.实验表明,基于 AFPTR 算法和预取量动态调节的预取机制,能够大幅降低用户端虚拟机的启动延迟,显著提高 COW 磁盘访问的本地命中率,从而为用户提高较好的虚拟机本地运行性能,支持虚拟机环境下按需的、快速的软件部署.

相同的 COW 磁盘可能会在网络环境下多个用户的机器上缓存.如果将这些分布在多个用户机器上的 COW 磁盘组织成一个 P2P 对等模式的虚拟磁盘映像共享系统,将有可能降低服务器压力,对进一步提高软件部署的效率以及分布环境下虚拟机迁移的效率等方面都具有重要的意义,这也是我们下一步即将开展的工作.

References:

[1] Smith JE, Nair R. The architecture of virtual machines. IEEE Computer, 2005,38(5):32−38.

[2] Sapuntzakis C, Chandra R, Pfaff B, Chow J, Lam M, Rosenblum M. Optimizing the migration of virtual computers. In: Proc. of the 5th Symp. on Operating Systems Design and Implementation. New York: ACM Press, 2002. 377−390. http://portal.acm.org/citation.cfm?id=844163

[3] Satyanarayanan M, Gilbert B, Toups M, Tolia N, Surie A, O'Hallaron DR, Wolbach A, Harkes J, Perrig A, Farber DJ, Kozuch MA, Helfrich CJ, Nath P, Lagar-Cavilla HA. Pervasive personal computing in an Internet suspend/resume system. IEEE Internet Computing, 2007,11(2):16−25.

[4] Chen B, Xiao N, Cai ZP, Wang ZY. An optimized COW block device driver in VMM for fast, on-demand software deployment. In: Proc. of the 1st IEEE/IFIP Int'l Workshop on End User Virtualization (EUV 2008). Washington: IEEE Computer Society, 2008. 387−392. http://portal.acm.org/citation.cfm?id=1489150

[5] Bellard F. QEMU, a fast and portable dynamic translator. In: Proc. of the USENIX Annual Technical Conf. (USENIX 2005). Berkeley: USENIX Association, 2005. 41−46. http://www.usenix.org/event/usenix05/tech/freenix/full_papers/bellard/bellard.pdf

[6] Chen PM, Noble DB. When virtual is better than real. In: Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS 2001). Washington: IEEE Computer Society, 2001. 133−138. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.74.9249&rep=rep1&type=pdf

[7] Chandra R, Zeldovich N, Sapuntzakis C, Lam MS. The collective: A cache-based system management architecture. In: Proc. of the 2nd Symp. on Networked Systems Design and Implementation (NSDI 2005). Berkeley: USENIX Association, 2005. 259−272. http://portal.acm.org/citation.cfm?id=1251203.1251222

[8] Pfaff B, Garfinkel T, Rosenblum M. Virtualization aware file systems: Getting beyond the limitations of virtual disks. In: Proc. of the 3rd Symp. on Networked Systems Design and Implementation (NSDI 2006). Berkeley: USENIX Association, 2006. 353−366. http://portal.acm.org/citation.cfm?id=1267706

[9] Reimer D, Thomas A, Ammons G, Mummert T, Alpern B, Bala V. Opening black boxes: Using semantic information to combat virtual machine image sprawl. In: Proc. of the 4th ACM/USENIX Conf. on Virtual Execution Environments (VEE 2008). New York: ACM Press, 2008. 111−120. http://portal.acm.org/citation.cfm?id=1346272

[10] Meyer DT, Aggarwal G, Cully B, Lefebvre G, Feeley MJ, Hutchinson NC, Warfield A. Parallax: Virtual disks for virtual machines. In: Proc. of the ACMSIGOPS/EuroSys European Conf. on Computer Systems 2008 (EuroSys 2008). New York: ACM Press, 2008. 41−54. http://portal.acm.org/citation.cfm?id=1352592.1352598

[11] Zhao M, Zhang J, Figueiredo R. Distributed file system virtualization techniques supporting on-demand virtual machine environments for grid computing. Cluster Computing, 2006,9:45−56. [doi: 10.1007/s10586-006-4896-x]

[12] Alpern B, Auerbach J, Bala V, Frauenhofer T, Mummert T, Pigott M. PDS: A virtual execution environment for software deployment. In: Proc. of the 1st ACM/USENIX Conf. on Virtual Execution Environments (VEE 2005). New York: ACM Press, 2005. 175−185. https://www.usenix.org/events/vee05/full_papers/p175-alpern.pdf

[13] Jin H, Liao XF. Virtualization of computing system: The great challenge in the area of computer architecture. Communication of CCF, 2008,4:15−23 (in Chinese).

[14] Liao XF, Jin H, Hu LT, Xiong XJ. LVD: A lightweighted virtual desktop management architecture. In: Proc. of the 2nd Int'l DMTF Academic Alliance Workshop on Systems and Virtualization Management: Standards and New Technologies (DMTF SVM 2008). Springer-Verlag, 2008. 25−36. http://www.springerlink.com/index/w25132508n48u1pp.pdf

[15] Zhang YH, Wang XL, Hong L. Portable desktop applications based on P2P transportation and virtualization. In: Proc. of the 22nd Large Installation System Administration Conf. (LISA 2008). Berkeley: USENIX Association, 2008. 133−144. http://www.usenix.org/event/lisa08/tech/full_papers/zhang/zhang.pdf

[16] Lu XC, Wang HM, Wang J. Internet-Based virtual computing environment (iVCE): Concepts and architecture. Science in China (Series E), 2006,36(10):1081−1099 (in Chinese with English abstract).

[17] Huai JP, Li Q, Hu CM. Research and design on hypervisor based virtual computing environment. Journal of Software, 2007,18(8): 2016−2026 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/2016.htm [doi: 10.1360/jos182016]

附中文参考文献:

[13] 金海,廖小飞.计算系统虚拟化:体系结构领域的重要挑战.中国计算机学会通讯,2008,4:15−23.

[16] 卢锡城,王怀民,王戟.虚拟计算环境 iVCE:概念与体系结构.中国科学(E 辑),2006,36(10):1081−1099.

[17] 怀进鹏,李沁,胡春明.基于虚拟机的虚拟计算环境研究与设计.软件学报,2007,18(8):2016−2026. http://www.jos.org.cn/1000-9825/18/2016.htm [doi: 10.1360/jos182016]

陈彬(1975－),男,湖北黄冈人,博士生,主要研究领域为虚拟化技术,分布式计算.

蔡志平(1975－),男,博士,副教授,CCF 高级会员,主要研究领域为虚拟化技术,网络安全.

肖侬(1969－),男,博士,教授,博士生导师,主要研究领域为网格计算,网络存储,体系结构,虚拟化技术.

王志英(1956－),男,博士,教授,博士生导师,主要研究领域为高性能计算机体系结构,虚拟化技术.