Journal of Computer Applications
计算机应用,2013,33(1):254-257,261
ISSN 1001-9081
CODEN JYIIDU
2013-01-01
http://www.joca.cn

文章编号:1001-9081(2013)01-0254-04
doi:10.3724/SP.J.1087.2013.00254

基于 Xen 虚拟机的内存资源实时监控与按需调整

胡　耀[1],肖如良[1*],姜　军[1],韩　佳[1],倪友聪[1],杜　欣[1],房丽娜[2]
(1.福建师范大学 软件学院,福州 350108;　2.深圳信息职业技术学院 软件学院,广东 深圳 518172)
(*通信作者电子邮箱 xiaoruliang@163.com)

摘　要:在虚拟计算环境中,难以实时地监控与分配内存资源。针对以上问题,基于 Xen 虚拟计算环境,提出一种能够实时监控 Xen 虚拟机内存(VMM)使用情况的 XMMC 方法并进行了实现。所提方法运用 Xen 虚拟机提供的超级调用,其不仅能实时地监控虚拟机内存使用情况,而且能实时动态按需分配虚拟机内存。实验结果表明,XMMC 方法对虚拟机应用程序造成的性能损失很小,低于5%;能够对客户虚拟机的内存资源占用情况进行实时的监测与按需调整,为多虚拟机的管理提供方便。

关键词:Xen;虚拟机内存;实时监控;按需调整
中图分类号:TP311.1　　**文献标志码**:A

Virtual machine memory of real-time monitoring and adjusting on-demand based on Xen virtual machine

HU Yao[1], XIAO Ruliang[1*], JIANG Jun[1], HAN Jia[1], NI Youcong[1], DU Xin[1], FANG Lina[2]
(1. *Faculty of Software, Fujian Normal University, Fuzhou Fujian* 350108, *China*;
2. *Department of Software, Shenzhen Institute of Information Technology, Shenzhen Guangdong* 518172, *China*)

Abstract: In a Virtual Machine (VM) computing environment, it is difficult to monitor and allocate the VM's memory in real-time. To overcome these shortcomings, a real-time method of monitoring and adjusting memory for Xen virtual machine called Xen Memory Monitor and Control (XMMC) was proposed and implemented. This method used hypercall of Xen, which could not only real-time monitor the VM's memory usage, but also dynamically real-time allocated the VM's memory by demand. The experimental results show that XMMC only causes a very small performance loss, less than 5%, to VM's applications. It can real-time monitor and adjust on demand VM's memory resource occupations, which provides convenience for the management of multiple virtual machines.

Key words: Xen; Virtual Machine Memory (VMM); real-time monitoring; adjusting on-demand

0　引言

随着云计算的发展,虚拟化技术的应用越来越广泛。虚拟化技术一个重要的应用就是服务器整合,通过在服务器上安装虚拟机(Virtual Machine, VM)可以达到整合分散的服务器资源,最大限度地提高服务器硬件资源的利用率,降低资源开销,减少企业运行成本。服务器整合通常通过在一台服务器上运行多层服务来实现,而不同层次的服务通常运行在不同的虚拟机中[1],此时这些虚拟机所需要的内存资源就各不相同。内存资源的多少是决定整个系统性能的关键因素之一,有些虚拟机可能需要更多的内存资源来处理更复杂的任务,而有些虚拟机则只需较少的内存资源就可以很好地提供服务。因此,为了更好地分配虚拟机的内存资源,提高内存资源的利用率,有必要对各个虚拟机内存(Virtual Machine Memory, VMM)资源的占用情况进行实时地监测与分析。

为了支持资源的监测,Xen 管理程序提供了监测控制工具来追踪包括 CPU、内存和磁盘等物理资源的使用情况。许多机构也在研究各种监测工具,包括 Gprof[2]、Oprofile[3]、Xenoprof[4] 和 XenMon[5]。虽然这些工具可以提供大量反映虚拟机运行情况的信息,但是它们在实时性方面有很大的限制,并且这些工具只工作在控制台模式,不易使用,更不能方便地调整系统内存资源。例如,XenMon,一款典型的监测工具,如果管理员在查看系统信息的同时需要实时地调整系统的内存资源,这款工具是不可行的[6]。

本文提出一种基于 Xen 虚拟机的内存资源实时监控(Xen Memory Monitor and Control, XMMC)方法。本文方法的目标是提供对虚拟机内存资源的实时监测,并对监测结果进行分析,为管理员实时调整虚拟机内存资源提供策略依据。通过对内存资源占用情况的实时监测,管理员可分析预测出某个虚拟机可能需要更多的内存资源来执行内存敏感的任务,以便能提供更好的服务,其可根据监测结果实时地为该虚拟机分配更多的内存资源;当内存敏感的任务执行结束后,又可回收部分空闲的内存资源,提高内存资源的利用率。

1　相关工作

对虚拟计算环境的资源使用实施监控已有很多相关的工作。文献[2]提出的 Gprof 是一款可执行程序调用图分析工具,其生成一个程序运行时详细的函数调用关系图,测量每个程序的处理时间。尽管 Gprof 需要占用大量的时间,因其频繁调用,但是其对测试优化很有用。为了构建正确的调用图,

收稿日期:2012-07-04;**修回日期**:2012-08-13。　**基金项目**:福建省科技计划重大项目(2011H6006)。
作者简介:胡耀(1987-),男,湖南永州人,硕士研究生,主要研究方向:虚拟化计算环境优化;　肖如良(1966-),男,湖南娄底人,教授,博士,CCF 高级会员,主要研究方向:系统虚拟化、云计算、软件工程;　姜军(1989-),男,福建三明人,硕士研究生,主要研究方向:云计算、系统虚拟化。

Gprof 用源代码的符号信息进行推断。文献[3]提出的 Oprofile 是一个 Linux 全系统分析器,能够以低开销分析所有运行的代码,它能监测各种硬件事件,如时钟周期、指令和缓存未命中等。Oprofile 由一个内核驱动,收集样本数据的守护进程和一些用来把数据转换成信息的后期分析工具组成。文献[4]提出的 Xenoprof 是一个带 Xen 特定驱动程序的 Oprofile 的扩展。Xenoprof 是为 Xen 虚拟机环境实现的全系统的统计监测工具,其把性能事件映射到特定的 H/W 性能计数器。为了在 Xen 环境中获取安装、停止和启动事件采样的 Hypercall,其支持全系统协调分析。Xenoprof 允许对并发执行的虚拟机和 Xen VMM 进行监测。文献[5]提出的 XenMon 运用 Xentrace 来实现信息的监测。Xentrace 是一款轻量级的事件记录工具,能够用来获取 Xen 中 trace buffer 的数据,Xen 中所有事件都被记录在 trace buffer 中。Xentrace 产生的记录数据量非常大,因此,Xen 提供了用户空间的处理工具 Xenbaked 来处理 trace buffer 中的新事件,并把这些记录转换成有意义的信息。经 Xenbaked 处理过的信息须经 XenMon 展示和记录。Xen 提供的 xm 程序是管理客户域的主要接口。xm 命令有很多选项,这些选项可以作为命令行参数被单独地调用,能够用来创建、暂停和关闭客户域,也能够用来列出当前客户域的信息[6]。

文献[7]提出了一种基于 Xen 虚拟机的实时工作负载监测反馈工具,其能够监测虚拟机上实时任务的执行,可以为 CPU 敏感的虚拟机提供更多的 CPU 资源。文献[8]提出了一种基于非托管语言的内存分配可视化工具,通过跟踪程序一段时间内运行的信息,动态模拟程序内存的分配过程。文献[9]提出了一个用于分布式环境下管理虚拟机完整性的监测系统,其通过引入特殊的虚拟机来实现对其他虚拟机的监测,然后通过 C/S 架构来传输信息。文献[10]提出了 Sandpiper——一款基于黑盒与灰盒的虚拟机资源管理工具,它能自动监测虚拟机的热点,决定物理资源到虚拟资源的映射,调整虚拟机适应新分配的资源,初始化虚拟机迁移所需的条件。Sandpiper 实现的黑盒方法是 OS 与应用程序完全无关的,灰盒方法用来统计 OS 与应用程序级的数据。文献[11]提出了一款虚拟机内存平衡器(Virtual Machine Memory Balancer, VMMB),由于其使用嵌套页面错误来监视内存访问,使用伪交换设备来监视客户的内存页面交换,增加了页面错误处理的开销,会导致虚拟机应用程序性能下降。文献[12]提出了基于最近最少使用(Least Recently Used, LRU)直方图的动态内存平衡方法,由于计算 LRU 直方图而引起的开销随页面数量而线性增加。因此,当负载的工作集很大时,其会引起严重的性能开销。

以上监控方法的缺点是:工作在控制台模式;缺乏对实时监控特性的支持。文献[2-6]的监控方法工作在控制台模式,操作不方便,在实时监控方面有很大的限制。文献[7-10]的监控方法虽然提供了友好的可视化工具,但是缺乏对实时监控特性的支持。本文提出的 XMMC 监控方法克服了以上工具的缺点,既支持实时监控的特性,又提供了友好的可视化界面。

2　XMMC 实时监控框架

本章主要描述 XMMC 实时监控方法的整体框架和部署框架以及各子模块的实现。XMMC 的部署架构如图 1,其部署在具有特权的虚拟机(Dom0)上,实时监测各个客户虚拟机(DomU)的内存资源占用情况。特权虚拟机控制着对物理资源的访问,能通过原生驱动直接访问物理资源。客户虚拟机不能直接访问物理资源,其对物理资源的访问必须经过特权虚拟机的协调,因此部署在特权虚拟机上能更有效地监测各个客户虚拟机的内存资源占用情况。XMMC 的整体系统架构如图 2,其不仅能够实时地监测虚拟机的内存占用情况,而且能够对虚拟机内存资源的分配进行控制和调整。Xen 提供了 xm 工具来实现虚拟机的日常管理,如关闭、暂停和启动等,但是该工具工作在控制台模式,很难做到实时地监测与调整虚拟机的内存资源。因此,本文通过用户界面实现一种结合了实时监测与控制的监控管理工具,其由以下两个功能模块组成。

1)内存监测模块。

该模块实时地监测客户虚拟机的内存资源占用情况,并以实时曲线的方式展现出来以便管理人员分析预测。

2)内存控制模块。

该模块可以对虚拟机的内存分配进行控制,管理人员可以根据监测模块的数据,按需地对虚拟机的内存分配进行调整。如果某个虚拟机的运行情况对内存敏感,则可以增加其内存来保证其运行的稳定性;否则可减少其内存来回收空闲的内存资源,提高内存资源利用率。

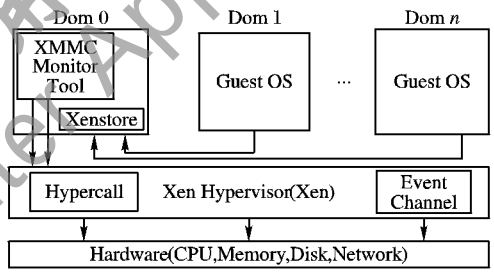

图 1　XMMC 部署架构

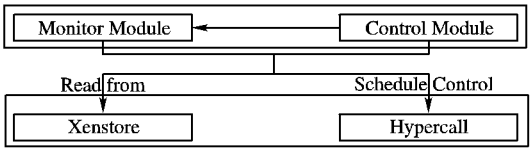

图 2　XMMC 系统架构

2.1　内存监测

该模块实时地监测客户虚拟机的内存资源占用情况,并以实时曲线的方式展现,其处理流程如图 3。为了实时地监测虚拟机的内存占用情况,首先通过 Xenstore 来实时地读取各个虚拟机的信息;然后根据读取的信息对每个虚拟机都调用 Xen 的 Hypercall 来监测其内存资源占用情况;最后把监测的内存占用情况用实时曲线展示出来。算法 1 展示了 XMMC 如何监测目标虚拟机的内存占用情况。其中,函数 get_domain_id()根据指定的虚拟机名字获取相应的虚拟机 id,其通过读取 Xenstore 中所有虚拟机信息,然后匹配目标虚拟机,并返回目标虚拟机的 id。函数 monitor_domain_memory()将监测指定 id 虚拟机的内存占用情况,其首先打开 Xen 提供的操作接口;其次调用 Xen 提供的 xc_domain_getinfo 函数来获取指定 id 虚拟机的信息,其中包括内存信息;然后,由于内存信息是按页存储的,每页的大小是 4 KB,需要对其进行适当的转换后可得到虚拟机的内存大小;最后,把获得的内存大小实时地展现出来。如图 4,该模块实时地展示了每个虚拟机的内存占用情况。纵坐标显示了内存占用率,横坐标显示了虚拟机的运行时间。该模块能够记录了虚拟机最近 60 s 的内存占用情况。

算法 1 虚拟机内存监控算法如下所示。

```
int get_domain_id(char *domain_name) {
  //定义 Xenstore 相关的结构体
  struct xs_handle *xsh = NULL;
  xs_transaction_t xth = XBT_NULL;
  //扫描 Xenstore 获取所有虚拟机的 id
  xsh = xs_domain_open();
  domain_ids = xs_directory(xsh, xth, "/local/domain", &size);
  //匹配目标虚拟机获取目标虚拟机的 id
  for(i = 0; i < size; i ++) {
    char *tmp[100];
    sprintf(tmp, "/local/domain/%s/name", domain_ids[i]);
    char *nameCandidate = xs_read(xsh, xth, tmp, NULL);
    //匹配成功返回目标虚拟机的 id
    if (0 == strncmp(domain_name, nameCandidate, 100)) {
      domain_id = atoi(domain_ids[i]);
      break;
    }
  }
  return domain_id;
}
void monitor_domain_memory(int domain_id, int *current_memory,
  int *max_memory) {
  //定义虚拟机操作句柄和虚拟机信息结构体
  xc_interface *xc_handle = NULL;
  xc_dominfo_t *domain_info = malloc(sizeof(xc_dominfo_t));
  xc_handle = xc_interface_open(NULL, NULL, 0);
  //获取虚拟机的信息
  result = xc_domain_getinfo(xc_handle, domain_id, 1, domain_
    info);
  //获取虚拟机的内存信息
  *current_memory = (domain_info->nr_pages * 4)/1 024;
  *max_memory = domain_info->max_memkb/1 024;
}
```

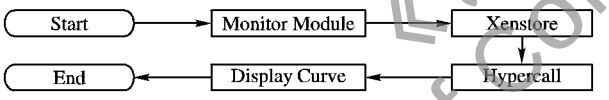

图 3 Monitor Module 处理流程

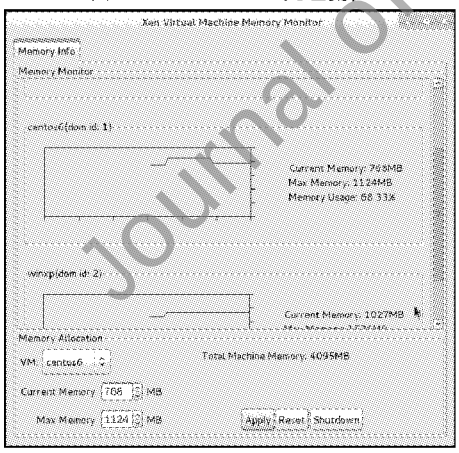

图 4 XMMC 实时曲线展示内存资源占用情况

2.2 内存控制

内存是决定系统整体性能的关键因素之一。内存的大小决定了虚拟机的运行情况,如果虚拟机上运行的是内存敏感的任务,为虚拟机分配过小的内存可能会导致该虚拟机不能很好地提供服务;如果虚拟机上运行的任务对内存不敏感,则可回收空闲的内存资源,避免资源的浪费。因此在实时监测虚拟机内存资源占用情况的同时,很有必要对监测的虚拟机进行内存资源的调整。该控制模块可以根据监测的内存使用情况,实时动态地调整各个虚拟机的内存分配情况,它首先通过 Xenstore 来读取需要控制的虚拟机信息;然后对该虚拟机调用 Hypercall 来调整其内存分配情况;最后实时地反映到监测模块上,即监测模块能实时地捕捉到这一调整行为,处理流程如图 5 所示。算法 2 展示了 XMMC 调整目标虚拟机内存的过程,其首先打开 Xen 提供的操作接口;其次调用 Xen 的 xc_domain_setmaxmem 来设置目标虚拟机的最大内存;然后调用 Xen 的 xc_domain_set_pod_target 来调整目标虚拟机的当前占用内存;最后,监测模块实时地监测到这一调整行为,并展现到实时曲线上。

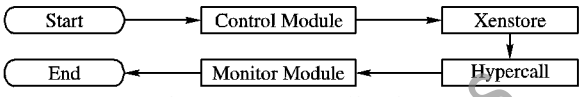

图 5 Control Module 处理流程

算法 2 虚拟机内存调整算法如下所示。

```
void set_domain_memory(int domain_id, int cur_memory, int max_
  memory) {
  xc_interface *xc_handle = NULL;
  xc_dominfo_t *info = malloc(sizeof(xc_dominfo_t));
  //虚拟机操作句柄
  xc_handle = xc_interface_open(NULL, NULL, 0);
  if (NULL == xc_handle) {
    return;
  }
  //设置指定虚拟机的内存
  xc_domain_setmaxmem(xc_handle, domain_id, max_memory *
    1 024);
  xc_domain_set_pod_target(xc_handle, domain_id, (cur_memory *
    1 024)/4, NULL, NULL, NULL);
  if (xc_handle)
    xc_interface_close(xc_handle);
  if (info)
    free(info);
}
```

3 实验评估

前面介绍了 XMMC 方法的架构以及子模块的实现,本章对 XMMC 进行相关实验与评估,本文主要关注 XMMC 对虚拟机带来的性能损失和 XMMC 监控方法的有效性两个方面。最终实验结果表明,XMMC 监控方法对虚拟机应用程序造成的性能损失很小,低于 5%;能够对客户虚拟机的内存资源占用情况进行实时地监测与按需调整。

3.1 实验环境

本实验采用 Xen 虚拟平台以及经过修改的 Linux 客户操作系统。实验环境如下:联想 ThinkVision QiTianM710E;CPU 为 Intel Core 2 Duo CPU E7500 2.93 GHz(3000 MHz) ×2;内存为 4 GB DDR3 PC3-10600 SDRAM 1333 MHz;硬盘为500 GB SATA 7200 RPM ST3500418AS;网卡为 Marvell Yukon 88E8057 PCI-E Gigabit Ethernet Controller;Xen 的版本为 4.1.2;特权虚拟机操作系统及内核为 Fedora 16 Linux-3.1.0-7.fc16.i686.PAE;Linux 客户虚拟机操作系统及内核为 CentOS 6 kernel-2.6.32;C/C++编译器为 GCC 4.6.2;Apache Web 服务器版本为 2.2.22。虚拟机配置一个虚拟 CPU(Virtual CPU,VCPU)。测试时除了特权虚拟机和受测试虚拟机运行外,无其他虚拟机运行。

首先,测试了 XMMC 实现对虚拟机性能的影响;然后,测试了 XMMC 监控方法的有效性。

3.2　性能损失测试

本文设计了两个实验,测试了 XMMC 本身对特权虚拟机和客户虚拟机上应用程序性能的影响。实验 1 的测试程序是常见的编译器 GCC 4.6.2,分别测试了编译 Apache-2.2.22 源代码的时间;实验 2 是自行设计的程序,该程序调用了 1 亿次 malloc 和 free 函数。最终实验结果如表 1 所示。

表 1　虚拟机中应用程序的运行时间对比　　s

实验	未部署 XMMC 监控方法		部署了 XMMC 监控方法	
	特权虚拟机	客户虚拟机	特权虚拟机	客户虚拟机
GCC 4.6.2	170.404	210.700	175.800	214.600
1 亿次调用 malloc 和 free	75.100	119.350	76.650	123.085

由表 1 可知,具有频繁内存申请和释放操作的 GCC 4.6.2程序,在正常的 Xen 虚拟机上编译一个应用程序所需的时间为:特权虚拟机需要 170.404 s,客户虚拟机需要 210.700 s;而在部署了 XMMC 监控方法的 Xen 虚拟机上编译同一个应用程序所需的时间为:特权虚拟机需要175.800 s,客户虚拟机需要 214.600 s。可见 XMMC 监控方法对特权虚拟机引起的性能损失在 5% 以内,对客户虚拟机引起的性能损失在 5% 以内。而对于自行设计的 1 亿次 malloc 和 free 函数调用的程序,XMMC 监控方法对特权虚拟机引起的性能损失在 1% 以内,对客户虚拟机引起的性能损失在 5% 以内,几乎可以忽略不计。

3.3　有效性测试

为了测试 XMMC 监控方法的有效性,本文选择开源的 Apache-2.2.22 Web 服务器。测试的目标是在 Xen 虚拟机环境下,实现在特权虚拟机(Dom0)上,对其他客户虚拟机的内存资源占用情况进行实时地监测与按需调整。测试方法是将 Apache Web 服务器部署在客户虚拟机上,然后模拟 10 万个客户端向该 Web 服务器提交 HTTP 请求,提交的 HTTP 请求会导致该 Web 服务器消耗的内存不断增加。当其内存消耗快逼近为该虚拟机分配的最大内存时,本方法对虚拟机的最大内存进行调整,增加其最大内存,以便能满足 Apache 对内存的需求。由于增加了虚拟机的最大内存,当客户端的 HTTP 请求结束后,Apache 消耗的内存会逐渐减少,为虚拟机分配的内存有大部分是空闲的,这就会导致内存资源的浪费。此时,再次对虚拟机的最大内存进行调整,减少其最大内存,回收部分空闲内存,提高内存资源的利用率。

如图 6 展示了本次测试的结果。图 6 中折线表示虚拟机当前内存,带符号的直线表示为该虚拟机分配的最大内存;其中:带三角符号的直线表示为该虚拟机分配的初始最大内存 1 204 MB;带正方形符号的直线表示当任务消耗的内存快逼近虚拟机初始最大内存时,为其分配的新的最大内存 1 536 MB;带菱形符号的直线表示任务结束后为虚拟机分配的最大内存 850 MB。从折线可知,随着客户端不断提交 HTTP 请求,Apache 消耗的内存也在不断增加,当消耗的内存 900 MB快逼近最大内存 1 024 MB 时,对虚拟机的最大内存进行调整,增加其最大内存为 1 536 MB,以便能满足 Apache 对内存的需求;当客户端的 HTTP 请求结束后,Apache 消耗的内存不断减少,由于之前对虚拟机增加了最大内存,这就会导致部分内存资源是空闲的,这时需要对这部分空闲的内存资源进行回收。此时,再次调整虚拟机的最大内存,减少其最大内存为 850 MB。

实验结果表明,XMMC 能实时地监测虚拟机的内存资源占用情况,当虚拟机上执行内存敏感的任务时,能实时地增大虚拟机的最大内存,以满足内存敏感任务对内存的需求;当内存敏感任务执行结束后,又能对空闲的内存资源进行回收。

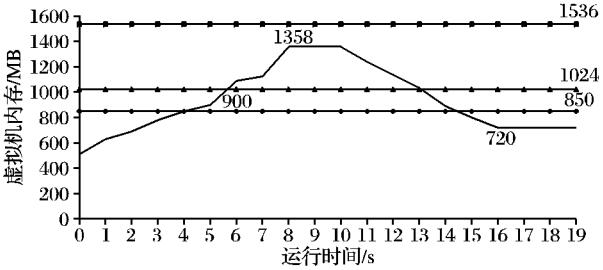

图 6　虚拟机内存资源实时监测与调整

4　结语

虚拟化技术在服务器整合方面的应用,提高了服务器硬件资源的利用率,降低了资源开销。内存资源是决定系统整体性能的关键因素之一。运行在服务器上的虚拟机内部署着各不相同的任务。本文提出一种实时监测虚拟机内存资源占用情况的 XMMC 监控方法,不仅能实时地监测虚拟机内存资源的占用情况,而且能够实时地根据系统实际的运行情况由管理员去调整虚拟机的内存分配,并用实时曲线展示了监测结果。通过实验表明,XMMC 方法具有很好的实时监测结果可视化,并可以实施动态调整分配虚拟机的内存资源,对内存不敏感的虚拟机内存资源进行回收。此项工作,为下一阶段进一步优化虚拟计算环境、提高系统资源的按需利用打下基础。

参考文献:

[1] 张建. Xen 虚拟机间通信优化研究与实现[D]. 上海: 上海交通大学, 2010.

[2] GRAHAM S L, KESSLER P B, McKUSICK M K. Gprof: a call graph execution profiler [C]// SIGPLAN'82: Proceedings of the 1982 SIGPLAN Symposium on Compiler Construction. New York: ACM Press, 1982: 120 - 126.

[3] LEVON J, ELIE P. Oprofile: a system profiler for Linux [EB/OL]. [2012-05-05]. http://oprofile.sf.net.

[4] MENON A, SANTOS R J, TURNER Y, *et al*. Diagnosing performance overheads in the Xen virtual machine environment [C]// VEE'05: Proceedings of the First ACM/USENIX International Conference on Virtual Execution Environments. New York: ACM Press, 2005: 13 - 23.

[5] GUPTA D, GARDNER R, CHERKASOVA L. XenMon: QoS monitoring and performance profiling tool [R]. California: Technical Report of HP Lab, 2005.

[6] Xen user's manual [EB/OL]. [2012-05-02]. http://oss.org.cn/ossdocs/server_storage/xen/user/user.html.

[7] KIM B K, JANG J H, HUR K W, *et al*. Monitoring and feedback tools for realtime workloads for Xen virtual machine [C]// Proceedings of the 2012 International Conference on IT Convergence and Security. Berlin: Springer, 2012: 151 - 161.

[8] ROBERTSON G G, CHILIMBI T, LEE B. AllocRay: memory allocation visualization for unmanaged languages [C]// Proceedings of the 5th International Symposium on Software Visualization. New York: ACM Press, 2010: 43 - 52.

(下转第 261 页)

试,传统方法所需脚本代码量为 6 384 行,本文方法所需代码量为 10 146 行;虽然对于同一个语言版本的测试来说,传统方法的代码量相对较少,但是却需要编写 13 种不同的语言版本,那么测试所需脚本代码总量即需要 8 万多行,而基于本文方法所设计的脚本只需要 1 种,测试代码总量为 1 万多行,从而大大减少了脚本冗余量,提高了测试效率。

3)当由于某种原因无法识别出控件本身的句柄时,也会造成测试不能正常进行,如图 6 所示。

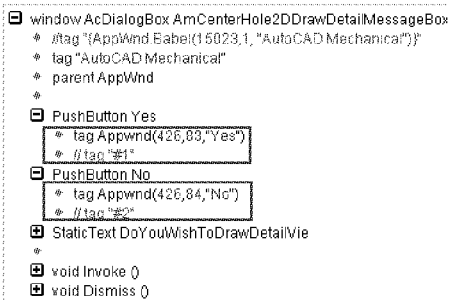

图 6　不能正常识别句柄的情况

图 6 中的 PushButton 类型的控件“Yes”和“No”不能根据有效的句柄识别出控件,只有“#”标识,这种情况下使用传统的基于句柄的方法是无法达到识别控件的目的,而在本文的方法中,利用如图所示的 tag Appwnd 的固定 ID 的方法,即 tag Appwnd(426,83,“Yes”)和 tag Appwnd(426,84,“No”)就可以有效地识别出此控件。

综上所述,基于本文所给出的利用 XML 与句柄相结合的 UI 识别方法测试 AutoCAD 产品对话框的方案,可以适用于全球化多语言版本的测试,并且能够确保 UI 自动化测试稳定的进行。

4　结语

UI 测试做得怎样,决定着软件产品质量的好坏,这是一项非常复杂的、需要有创造性的工作。本文针对 AutoCAD 产品多语言多版本的特点,提出一种基于 SilkTest 的句柄和 XML 相结合的 UI 控件识别方法,并在此方法的基础上给出了对 AutoCAD 系列产品对话框的自动化测试方案。该方法可以有效避免由于句柄出现异常变化导致自动化测试无法正常进行的情况,同时也在一定程度上提高了 UI 识别的稳定性,确保对 AutoCAD 产品的 UI 测试的正常进行。本文主要关注了对 UI 控件的识别,而对抽取出的信息处理考虑较少,这将是下一步工作的重点。

参考文献:

[1] MU B, ZHAN M K, HU L F. Design and implementation of GUI automated testing framework based on XML [C]// Proceedings of the 2009 WRI World Congress on Software Engineering. Washington, DC: IEEE Computer Society, 2009: 194 – 199.

[2] GANOV S R, KILLMAR C, KHURSHID S, *et al*. Test generation for graphical user interfaces based on symbolic execution [C]// Proceedings of the 3rd International Workshop on Automation of Software Test. New York: ACM Press, 2008: 33 – 40.

[3] MEMON A M. Automatically repairing event sequence-based GUI test suites for regression testing [J]. ACM Transactions on Software Engineering and Methodology, 2008, 18(2): 132 – 138.

[4] MEMON A M, PLOOACK M E, SOFFA M L. Hierarchical GUI test case generation using automated planning [J]. IEEE Transactions on Software Engineering, 2001, 27(2): 144 – 155.

[5] XIE Q. Developing cost-effective model-based techniques for GUI testing [C]// Proceedings of the 28th International Conference on Software Engineering. New York: ACM Press, 2006: 997 – 1000.

[6] 王钊, 白晓颖, 戴桂兰. 基于有色 Petri 网模型的 GUI 测试用例自动生成技术[J]. 清华大学学报: 自然科学版, 2008, 48(4): 601 – 604.

[7] 李小将. 基于 CIS 的消息驱动的 GUI 自动测试方法[J]. 计算机工程, 2004, 30(5): 12 – 14.

[8] BAI X Y, LU H, ZHANG Y, *et al*. Interface-based automated testing for open software architecture [C]// COMPSACW 2011: Proceedings of the 35th IEEE Annual Computer Software and Applications Conference. Washington, DC: IEEE Computer Society, 2011: 149 – 154.

[9] 朱芳, 李曦, 赵振西. 一种多平台自动化测试工具的设计和实现[J]. 计算机工程, 2004, 30(24): 186 – 188.

[10] BERTOLINO A, GAO J H, MARCHETTI E, *et al*. TAXI—a tool for XML-based testing [C]// Proceedings of the 29th International Conference on Software Engineering-Companion. Washington, DC: IEEE Computer Society, 2007, 26(6): 53 – 54.

[11] 杨怡君, 黄大庆. Android 手机自动化性能测试工具的研究与开发[J]. 计算机应用, 2012, 32(2): 554 – 556.

[12] SUN Y X, CHEN H Y, TSE T H. Lean implementations of software testing tools using XML representations of source codes [C]// 2008 International Conference on Computer Science and Software Engineering. Washington, DC: IEEE Computer Society, 2008: 708 – 711.

[13] 安金霞, 王国庆, 李树芳, 等. 基于多维度覆盖率的软件测试动态评价方法[J]. 软件学报, 2010, 21(9): 2135 – 2147.

[14] XING B, GAO L, ZHANG J, *et al*. Design and implementation of an XML-based penetration testing system [C]// 2010 International Symposium on Intelligence Information Processing and Trusted Computing. Washington, DC: IEEE Computer Society, 2010: 224 – 229.

[15] KIM-PARK D S, de la RIVA C, TUYA J. An automated test oracle for XML processing programs [C]// Proceedings of the First International Workshop on Software Test Output Validation. New York: ACM Press, 2010: 5 – 12.

[16] MORRIS K C. A framework for XML schema naming and design rules development tools [J]. Computer Standards and Interfaces, 2010, 32(4): 179 – 184.

(上接第 257 页)

[9] FANG H F, ZHAO Y Q, ZANG H Y, *et al*. VMGuard: an integrity monitoring system for management virtual machines [C]// Proceedings of the 16th IEEE International Conference on Parallel and Distributed Systems. Piscataway: IEEE Press, 2010: 67 – 74.

[10] WOOD T, SHENOY P, VENKATARAMANI A, *et al*. Sandpiper: black-box and gray-box resource management for virtual machines [J]. Computer Networks, 2009, 53(17): 2923 – 2938.

[11] MIN C, KIM I, KIM T, *et al*. VMMB: virtual machine memory balancing for unmodified operating systems [J]. Journal of Grid Computing, 2012, 10(1): 69 – 84.

[12] ZHAO W M, WANG Z L. Dynamic memory balancing for virtual machines[C]// VEE'09: Proceedings of the Fifth ACM/USENIX International Conference on Virtual Execution Environments. New York: ACM Press, 2009: 21 – 20.