软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(6):1620−1634 [doi: 10.3724/SP.J.1001.2012.04118]

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

虚拟机确定性执行重放的模型分析和实现方法*

于佳耕[1,2+], 周 鹏[1], 武延军[1], 赵 琛[1]

[1](中国科学院 软件研究所 基础软件国家工程研究中心,北京 100190)

[2](中国科学院 研究生院,北京 100049)

Model Analysis and Implementation Method of Deterministic Execution Replay Based on Virtual Machine

YU Jia-Geng[1,2+], ZHOU Peng[1], WU Yan-Jun[1], ZHAO Chen[1]

[1](National Engineering Research Center of Fundamental Software, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: E-mail: yujiageng734@gmail.com

Yu JG, Zhou P, Wu YJ, Zhao C. Model analysis and implementation method of deterministic execution replay based on virtual machine. *Journal of Software*, 2012,23(6):1620−1634. http://www.jos.org.cn/1000-9825/4118.htm

Abstract: To make the replay deterministic, the study presents the definition of VM replay by constructing a VM execution model, and then proves the sufficient conditions of VM replay using formal expressions of the algebra. Based on these conditions, the paper presents CASMotion, a Xen based implementation of VM execution replay. CASMotion classifies the category of non-deterministic events in Xen domU and presents their replaying methods and time matching algorithms. The experiment results show CASMotion can accurately replay the non-deterministic events with low performance penalty.

Key words: execution replay; non-deterministic event; model analysis; virtual machine; Xen

摘 要: 进程重放用于程序调试,无法重现系统全部状态,难以分析错误根源.而系统级重放复杂且难于实现,尚无模型分析方法提供理论指导,确保重放执行与记录执行等价.为了使执行重放系统适用于系统调试,建立虚拟机指令执行模型,提出了虚拟机执行重放的定义,给出并证明了成功重放的充分条件.根据该充分条件,设计实现了基于Xen的虚拟机重放系统CASMotion.CASMotion讨论了Xen DomU中不确定事件的种类,给出各类事件的重放方法以及时间点的匹配算法.CASMotion成功实现了不确定事件的准确重放,实验结果表明其具有较低的性能损失.

关键词: 执行重放;不确定事件;模型分析;虚拟机;Xen

中图法分类号: TP316 **文献标识码**: A

执行重放[1]是通过记录程序在一次运行时的各种不确定状态,并在下一次执行时根据记录重放整个执行过程的方法.执行重放在实现上有顺序重放和内容重放两类执行重放策略.顺序重放给选取的操作添加标记,以

* 基金项目: 国家自然科学基金(90818012); 中国科学院知识创新工程(KGCX2-YW-125)
收稿时间: 2010-11-12; 定稿时间: 2011-08-24

记录操作间的偏序关系,重放时根据偏序关系确保操作按原来的顺序发生,从而保证重放过程与初始执行过程基本等价.而内容重放是指在初次执行时,记录进程或系统与其运行环境的所有输入和输出的内容,重放时,该进程或系统的所有输入都从记录中读取.内容重放策略的缺点是性能开销较大,而且难以实现和维护.所以,目前研究的执行重放技术是针对顺序重放策略而言的.

除了上述重放策略的差异外,不同的应用方向对重放的精确度要求也不同.例如,虚拟机入侵检测方向只关心网络交互,重放系统透明地记录攻击的发生过程,并在随后按日志重放网络攻击.该领域只需要重放与攻击相关事件,不需对全系统重放;迁移重放是将日志传输至目的虚拟机,通过重新执行日志中的事件完成虚拟机迁移.由于迁移重放多用于提供网络服务的虚拟环境,要求目的虚拟机和源虚拟机提供相同的对外服务,即它们外部观察等价,而对虚拟机系统迁移前后的内部状态无要求;虚拟机调试方向则是使用重放技术重现客户虚拟机系统及其上运行程序的不确定性错误,而使用传统的循环调试方法难以重现和分析这类不确定错误.对于以完成虚拟机调试为目标的重放系统,不确定错误可能发生在系统运行的各个状态,所以只有在重放阶段能够模拟出记录阶段的所有虚拟机状态,才能精确地重现错误.由此可知,虚拟机调试对重放系统要求最高、最具难度.但目前已有的研究工作还未能从模型分析的角度给出适用于调试场景的重放过程形式化描述方法,进而无法说明完成的系统是否具有准确重放虚拟机系统内部状态的能力.

本文针对虚拟机的系统调试对执行重放技术的需求,扩充了指令集虚拟化模型,提出了虚拟机系统执行重放的形式化定义,讨论不确定性事件与达成执行重放的关系,并证明了完成系统重放的充分条件.即证明所有达成该充分条件的前提下,通过重放指令序列中的不确定性指令,可以使得虚拟机重放系统能够准确模拟记录阶段的执行状态,进而使得重放系统具有完成系统调试的能力.本文依据重放充分条件在 Xen 虚拟机监控器上设计与实现了 CASMotion 重放系统,并使用较新的 Intel Core2 处理器实现硬件辅助功能.最后给出实验评估和讨论,并与相关工作进行比较.

1 Xen 虚拟机介绍

本文以 Xen 3.3.0 类虚拟化系统[2]作为实现虚拟机系统重放的实验平台.类虚拟化技术通过修改部分操作系统的代码,使得操作系统与虚拟机管理器相配合,实现系统虚拟化.图 1 是 Xen 系统的整体结构图.虚线框围住的部分表示虚拟化层的范围,它是由一个虚拟机管理器和一个特权虚拟机组成.

在 Xen 中,每一个系统就是一个域(domain).图 1 中特权域称为 Domain0(或 Dom0),而其他虚拟机被称为 DomainU(或者 DomU).从虚拟机到 Xen 虚拟机监控器的系统调用称为超级调用(hypercall).

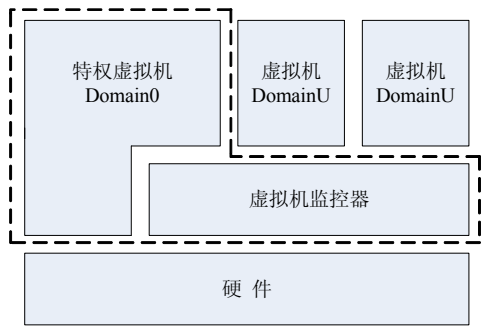

Fig.1 Architecture of Xen

图 1 Xen 整体结构

在虚拟环境下,一个虚拟机不会直接收到来自硬件的外部中断,只会收到虚拟机监控器注入的虚拟中断.根据中断的来源分类,Xen 之上的虚拟机可能收到的外部中断有 4 种:来自物理设备发起的外部中断、来自虚拟机监控器、来自同一个虚拟机其他 VCPU 以及来自其他虚拟机.在 Xen 中,事件是包括中断在内的异步消息触发

的抽象.Xen 实现了事件通道(event channel)作为虚拟机监控器通知客户虚拟机某一事件发生的通道.中断处理是 Xen 事件通道的一种应用.

Xen 使用了前后端模式实现类虚拟化中的外部设备虚拟化.除了 Dom0 和其他的隔离驱动域外,非特权域(DomU)并不拥有真正的物理设备,而只拥有虚拟设备.Xen 虚拟设备在初始化时向 XenBus 注册,而将大部分的初始化和设置推迟到 XenBus 探测(probe)时.Xen 常规块设备驱动程序分为前端驱动(front-end driver)和后端驱动(back-end driver).当前端设备驱动接到来自于客户操作系统的请求时,通过事件通道、授权表和环缓冲区向隔离驱动域的后端设备提出服务请求.后端设备收到服务请求后,通过本地(native)设备完成读写请求.XenBus 上典型的虚拟设备包括虚拟块设备(VDB)、虚拟网络接口(VNIF)等.

Xenstore 是 Xen 提供的一个域间共享的存储系统,基于共享内存页与事件通道实现虚拟机间的通信.Xenstore 中存储了各个虚拟机(包括 Dom0)的配置信息,例如 Domain ID、Domain Name、UUID、前后端设备、启动时间、虚拟机状态等.

2 虚拟机执行重放模型分析

本节首先参考了系统虚拟机指令状态表达模型(见第 2.1 节),并在该模型基础上针对虚拟机调试应用的要求,提出虚拟机执行重放的定义(见第 2.2 节).证明了完成系统重放的充分条件(见第 2.3 节),最后总结实现准确重放的策略(见第 2.4 节).

2.1 系统虚拟机指令状态模型

本节根据文献[3]中的虚拟机指令状态表达模型.该模型可表达虚拟执行环境中的指令集虚拟化特性,适用于表达关注指令执行顺序的虚拟机执行重放系统.

定义 1(虚拟机状态表达). 定义任意时刻虚拟机的执行状态为

$$S=\langle E,M,P,R\rangle,$$

其中,E 为可执行指令序列,其中存储了当前虚拟机中所有的可执行单元.$E[i],0\leqslant i<q$ 表示其中一个可执行单元,设定虚拟机监控器中可执行单元的总数为 q.M 表示为该虚拟机服务的处理器的特权级,user 态或 supervisor 态.当 M=supervisor 时,表示虚拟机正陷入到虚拟机监控器中执行.P 表示虚拟机将要执行的操作,$P=E[i],0\leqslant i<q$.R 表示当前虚拟机能够执行的指令的范围,超出该范围的指令需要“陷入再模拟”执行.$R=(l,b)$,l 为指令起始的绝对机器地址,b(bound)为指令的界限.即将指令集空间$(l,l+b)$映射给虚拟机使用,虚拟机执行该区间的指令时,不需要陷入到虚拟机监控器中.

$R=(l,b)$是为完成指令集虚拟化而设置的重定向限制寄存器.设虚拟机中执行任意指令 x,$f(x)=l+x$ 为其在宿主机指令集上对应的指令,也是为该虚拟机服务的 CPU 实际执行的指令.由虚拟机状态表达模型,可得

$$\begin{cases} l\leqslant f(x)<l+b, & \text{非敏感指令}\\ l+b\leqslant f(x)<q, & \text{敏感指令}\\ f(x)\geqslant q, & \text{非法敏感指令}\end{cases}.$$

根据上述对敏感和非敏感指令的界定,给出虚拟机陷入到虚拟机监控器的条件.

定义 2(陷入再模拟触发条件). 当客户机执行指令 a 时,下列的情况会使得客户机陷入到虚拟机监控器中:

$$\begin{cases} a+l\geqslant q, & (1)\\ a\geqslant b, & (2)\end{cases}$$

情况(1)超出实际指令集的范围,情况(2)表示执行了敏感指令,两者都需要陷入到虚拟机监控器中.

对于虚拟客户机,其状态 S 构成一个可数有限集和,定义为 C.虚拟机一次执行的指令序列为 I.

定义 3(状态迁移). 定义指令 $i\in I$ 为 C 到 C 的状态迁移,即 $i:C\to C$.

例如,$i(S_1)=S_2$ 或者 $i(E_1,M_1,P_1,R_1)=(E_2,M_2,P_2,R_2)$.

如果指令 $i\in I$ 表示陷入过程,即 $i(E_1,M_1,P_1,R_1)=(E_2,M_2,P_2,R_2)$,那么会进行如下操作:

1. 指令复制:$E_2[j]=E_1[j]$, for $0\leqslant j<q$;
2. 保存执行环境:$E_2[0]=(M_1,P_1,R_1)$;
3. 重置执行环境:$(M_2,P_2,R_2)=E_1[1]$;

显然,$M_2=supervisor$,$R_2=(0,q-1)$.

2.2 虚拟机系统重放定义

本节在虚拟机指令状态表达模型的基础上,提出指令级虚拟机重放的定义.

重放系统的基本假设:

假设 1. 记录阶段和重放阶段,两个 VM 的初始状态一致.

假设 2. 记录阶段和重放阶段,VM 与外部交互获得的内容一致.

基于以上两个假设,下面的讨论只需关心执行的顺序问题.

前文已定义 C 为虚拟机状态 S 的集合,并定义指令为其上的状态迁移,并且 C 对状态迁移闭包.所以,任意形如 $e_{(i,k)}(S_i)=k\ldots ji(S_i)=S_k$ 的指令序列可被看为 C 上的状态迁移序列,$e_{(i,k)}$表示从状态 S_i 迁移到 S_k 的执行序列.虚拟机一次执行的指令序列为 I,且$\forall e\subset I$,令 $e_{(i,i)}S_i=S_i$.

定义虚拟机的一次执行为 $X=\langle C,I\rangle$,C 为虚拟机在执行中的状态集合,I 为虚拟机该次执行的指令序列.

定义 4(虚拟机重放映射). 对$\forall S_i\in C_{record}$ 和以 S_i 为起始状态的任意执行序列 $e_{(i,j)}\subset I_{record}$,存在执行序列 $e'_{(i,j)}\subset I_{replay}$ 使得 $f(e_{(i,j)}(S_i))=e'_{(i,j)}(f(S_i))$,即两个集合间存在同态映射 $f:X_{record}\rightarrow X_{replay}$.

定义 5(虚拟机重放). 若虚拟机的两次执行 X 与 X'存在虚拟机映射 $f:X\rightarrow X'$,且满足基本假设条件,则称 X' 为 X 的执行重放.

综上,虚拟机系统重放的充分条件是:对记录阶段的任意状态$\forall S_i\in C_{record}$ 以及其上的任意执行序列 $e_{(i,j)}\subset I_{record}$,在重放阶段都可以找到与之相对应的 $e'_{(i,j)}\subset I_{replay}$ 使得 $f(e_{(i,j)}(S_i))=e'_{(i,j)}(f(S_i))$ 成立.记录阶段的虚拟机状态都可以同态映射到重放阶段的虚拟机状态,即在重放阶段可以模拟记录阶段的所有执行.

2.3 不确定性事件与虚拟机重放

PWD 假设[4]认为:对进程而言,非确定性事件发生后到下一个非确定性事件发生前,进程的运行状态是处于确定状态的.同样,对于虚拟机系统,DomU 可看作是运行在真实物理机器之上的进程.所以,PWD 假设也适用于虚拟机系统.就单核虚拟机来说,非确定性事件又为虚拟机的外部交互所产生.据此,本文提出引理 1,作为研究不确定性事件和虚拟机重放关系的基础.本部分描述执行确定性的含义,并证明不确定性事件以一定的方式重放后可达成虚拟机系统重放,即证明虚拟机系统重放的充分条件.

引理 1. 设单核虚拟机的任意两次无外部交互的执行为 X 与 X',且它们的初始状态相同,即 $S_0=S'_0$.那么存在一一映射σ,使得$\sigma:X\rightarrow X'$且$\sigma^{-1}:X'\rightarrow X$ 成立,X 与 X'能够互相模拟对方的指令执行,记为 $X\cong X'$.

进一步来说,对$\forall S_i\in C$ 和以 S_i 为起始状态的任意执行序列 $e_{(i,j)}\subset I$,存在执行序列 $e'_{(i,j)}\subset I'$ 使得$\sigma(e_{(i,j)}(S_i))=e'_{(i,j)}(\sigma(S_i))$,并且反之亦成立,即对 $\forall S'_i\in C'$ 和以 S'_i 为起始状态的任意执行序列 $e'_{(i,j)}\subset I'$,存在执行序列 $e_{(i,j)}\subset I$ 使得 $\sigma^{-1}(e'_{(i,j)}(S'_i))=e_{(i,j)}(\sigma^{-1}(S'_i))$.

解释 1. 设单核虚拟机的任意两次无外部交互的执行为 X 与 X',且它们的初始状态相同即 $S_0=S'_0$,并且 X 与 X'具有相同的长度指令序列,那么自然得到 $X\cong X'$.

解释 2. 设单核虚拟机的任意两次执行 X 与 X',在各自执行了 k 次外部交互后具有相同的状态,即 $S_{ND_k}=S'_{ND_k}$.那么在下次外部交互发生之前,以 S_{ND_k},S'_{ND_k} 为起始状态的相同长度的执行序列 $S_{ND_k}\cong S'_{ND_k}$.

设虚拟机一次执行X中记录的非确定性指令执行序列为 I_{ND},不涉及外部交互的指令为确定性指令,记为 I_D. $I=I_{ND}\cup I_D$ 并且 $I_{ND}\cap I_D=\varnothing$.

命题 1. 满足第 2.2 节的基本假设前提下,在 X'中以同样的顺序重新执行 X 的 I_{ND},如果重放阶段任意相邻指令$\forall i,j\in I_{ND}$ 间的指令集合 $I'_{(i,j)}\subset I'_D$ 与记录阶段的相应指令集合 $I_{(i,j)}\subset I_D$ 存在关系 $|I'_{(i,j)}|=|I_{(i,j)}|$,并且对任意指令$\forall i\in I_D$ 在 X'中被模拟执行后虚拟机状态 $S'_i=S_i$,那么一定存在虚拟机重放映射 $f:X\rightarrow X'$,即 X'为 X 的执行重放.

证明:假设 X 中存在单指令 i,则 $i\in I\Rightarrow i\in I_{ND}$ 或者 $i\in I_D$.设 i 是第 1 个使得虚拟机重放映射不成立的指令.即存在同态映射 f 使得对 $\forall S_k,0\leqslant k<i$ 有 $f(e_{(k,i-1)}(S_k))=e'_{(k,i-1)}(f(S_k))$,但 $f(e_{(k,i)}(S_k))\neq e'_{(k,i)}(f(S_k))$.

1) 首先,若 $i\in I_D$

构造映射 g

$$g=\begin{cases}g(S_k)=f(S_k), & 0\leqslant k<i\\ g(S_k)=S_k, & k=i\end{cases}.$$

对 $\forall S_k,0\leqslant k<i,g(e_{(k,i)}S_k)=g(S_i)=S_i$ 并且 $e'_{(k,i)}g(S_k)=e'_{(k,i)}S'_k=S'_i$.由题设可知,对 $i\in I_{ND}$,有 $S'_i=S_i$.可得 $g(e_{(k,i)}S_k)=e'_{(k,i)}g(S_k)$,映射 g 为虚拟机重放映射,所以必然有 $i\notin I_{ND}$.

2) 若 $i\in I_D$

令 $nd_m<i<nd_{m+1}$,且 $nd_m,nd_{m+1}\in I_{ND}$.

当 m=0 时,nd_1 为第 1 个执行的非确定性指令,并且 $|I'_{(0,1)}|=|I_{(0,1)}|$,由引理 1 的解释 1 可知,虚拟机的执行状态是确定的,即对 $\forall S_k,0\leqslant k<i$,有 $f(e_{(k,i)}(S_k))=e'_{(k,i)}(f(S_k))$.当 m>0 时,由于 i 是第 1 个使得虚拟机重放映射不成立的指令,又因为 $S_{nd_m}=S'_{nd_m}$,并且 $|I'_{(nd_m,nd_{m+1})}|=|I_{(nd_m,nd_{m+1})}|$,由引理 1 的解释 2 可知,直到 nd_{m+1} 执行之前,虚拟机的执行状态是确定的.即对 $\forall S_k,nd_m\leqslant k<i$,有 $f(e_{(k,i)}(S_k))=e'_{(k,i)}(f(S_k))$.所以必然有 $i\notin I_D$.

综上,不存在满足题设的指令 i,必然存在虚拟机重放映射 $f:X\to X'$,即 X' 为 X 的执行重放. □

由上可得,在满足第 2.2 节基本假设前提下,单核虚拟机重放的充分条件为:

充分条件 1. 重放阶段任意相邻指令 $\forall i,j\in I_{ND}$ 间的指令集合 $I'_{(i,j)}\subset I'_D$ 与记录阶段的相应指令集合 $I_{(i,j)}\subset I_D$ 存在关系 $|I'_{(i,j)}|=|I_{(i,j)}|$.

充分条件 2. 任意指令 $\forall i\in I_{ND}$ 在 X' 中被模拟执行后使得虚拟机的状态与记录时一致,即 $S'_i=S_i$.

满足上述条件即可进行成功的单核虚拟机系统重放.

2.4 虚拟机系统重放策略

假设 1 要求重放执行 X' 与记录执行 X 起始于相同的系统检查点(check point),这保证了 $S'_0=S_0$.在单核虚拟机环境下,非确定事件是通过与外部交互触发的.根据非确定性事件的触发方,可将非确定事件分为两类:

- 由虚拟机本身发起读取系统平台信息的事件,比如 RDTSC,cpuid 等.该类事件称为数据不确定事件;
- 由外部系统主动通知虚拟机的事件,比如设备输入、外部中断等.该类事件称为时间不确定事件.

本研究使用一种实现充分条件 1 的方案,即在重放阶段将非确定性指令 $\forall i\in I_{ND}$ 插入到与记录阶段相同的执行序列位置,从而使得记录阶段的两两非确定性指令之间的确定性指令可在重放阶段找到与其一一对应的指令.为了将不确定事件相关指令正确地插入到指令序列中,本研究使用程序性能计数器辅助的方式记录“时间点”.每个不确定性事件附带“时间点”标记由三元组⟨*bc*,*eip*,*ecx*⟩构成.*bc* 表示自启动记录模式以来受监控虚拟机上分支计数器的累加值,两个相邻不确定性事件的 *bc* 差值表示从前一个事件到下一个事件发生的分支次数.*eip* 指虚拟机当前的指令指针,*ecx* 用于标识 rep 前缀操作指令执行次数.由于虚拟机内的程序有时可反复调用多次,因而只用 *eip*,*ecx* 无法定位指令序列的位置.使用 *bc* 确定在某程序分支下,匹配 *eip*,*ecx* 就可在重放阶段精确定位指令插入位置.对于数据不确定事件,只需等其发生时按照⟨*bc*,*eip*,*ecx*⟩寻找相关输入内容重放即可;对于时间不确定事件,则需要在重放阶段按照⟨*bc*,*eip*,*ecx*⟩插入相关指令.

假设 2 要求在重放不确定性事件时,回填与记录阶段相同的数据内容.再结合具体事件的重放策略,联合保证充分条件 2 的成立.

3 虚拟机重放的方法

根据第 2 节的理论分析,只要在满足第 2.2 节基本假设前提下,保证充分条件 1 和充分条件 2 的成立,就可以解决虚拟机非确定事件的精确重放.在这一理论的指导下,我们设计并实现了 CASMotion 虚拟机重放系统.

首先讨论重放事件的类别,接着给出实现精确重放必须的“时间点”标记的实现方法,最后分别论述记录和重放的方法.

3.1 Xen虚拟机重放的事件分类讨论

前文已经得出需重放的不确定事件包含数据和时间不确定性两类事件,本节结合 Xen 系统讨论上述两类事件具体的范畴,并据此给出一般虚拟机重放事件的分类讨论.

首先,CASMotion 不需要重放外部交互的全过程,只需重放引起虚拟机处于不确定状态的事件.比如虚拟机和网络设备的交互,其中,从 Dom0 的真实网络设备中读取数据的事件为不确定性事件,而虚拟机通过网络设备向外部发送数据包的事件则是确定性的事件.同样的,虚拟机内存和磁盘的操作也不属于不确定事件.经过重放阶段的起始检查点恢复后,虚拟机内存和磁盘被恢复到与记录阶段相同的状态.在之后的重放过程中,对内存和磁盘的修改或是由系统固定周期的写入或是受不确定性事件的影响而写入,所以 CASMotion 不需对内存和磁盘操作进行重放.

数据不确定事件是指由 DomU 发起的请求,但随着物理平台的不同以及时间的推移,返回结果不确定的事件.该类事件会导致虚拟机陷入到 Xen 中,随后被模拟执行.按照陷入到 Xen 的方式又可分为两类:通过超级调用(hypercall)陷入或者通过通用保护(GP)机制陷入.以 Xen3.3.0 版本的 37 个超级调用为例,去掉特权域使用的域管理调用、内存管理调用以及虚拟机申请和设置本机资源调用,剩余 3 个可看作是备选的数据不确定事件:读取调式寄存器(do_get_debugreg)、读取 Xen 版本信息(do_xen_version)和读取控制台数据(do_console_io).由于客户虚拟机从 VCPU 数据结构中读取调式寄存器信息,不直接读取硬件寄存器,可排除该超级调用;本研究在记录和重放实验中使用相同的物理平台和 Xen 版本,保证读取到 Xen 版本信息是一致的,也可排除 do_xen_version 调用;依据后文叙述的设备重放方法,已在更底层的结构中重放了控制台的输入数据和中断通知,超级调用 do_console_io 也不属于数据不确定事件.虚拟机在执行敏感指令,并且该指令没有被超级调用取代时,就会通过通用保护(GP)机制陷入到 Xen 中.在通常情况下,为了维护各个域的独立性模拟执行的敏感指令不会直接操控硬件,但在某些特殊情况下会通过设置使得敏感指令直接读取真实的系统信息,比如读取时间戳记数器(RDTSC)和获得 CPU 信息(cpuid).在后文中,以 RDTSC 为例给出记录和重放数据不确定事件的算法.所以,针对数据不确定事件,只需重放一些特殊的操作,如 RDTSC,cpuid 等.

时间不确定事件是指由虚拟机监控器和特权域向受监控虚拟机发起的请求,在重放阶段需要重放出事件发生的时间和内容.本研究暂不考虑 DomU 之间的通信,而只关注受监控 DomU 和 Dom0 间的交互.按时间不确定事件的功能划分,可分为两种类型:系统状态更新事件和设备输入事件.前者影响的范围是 Xen 和 DomU 共享的重要数据结构,如 VCPU 控制结构、共享信息页(shared_info)以及启动信息页(start_info).具体的事件包括 VCPU 的调度、墙上时间和 VCPU 时钟更新.设备输入事件方面包含一般虚拟机使用的控制台设备、网络设备和 USB 设备.

本文分别以 RDTSC 事件和控制台输入事件为例,给出 CASMotion 系统中数据不确定事件和时间不确定事件的记录、重放方法,并论述时间点的获取和匹配算法.其中,控制台输入事件的重放是由事件通道(event channel)通知事件驱动.事件通道(event channel)通知事件属于时间不确定事件,而其记录方法类似于 RDTSC 事件,本文将在第 3.4 节给出其重放方法.

3.2 获取时间点的方法

时间点为插入指令提供准确的指令序列位置标识.如前文所述,在 X86 系统上需要用⟨*bc*,*eip*,*ecx*⟩三元组唯一地表示一个时间点.其中,EIP 和 ECX 寄存器的获取比较容易,而 BC 则相对比较困难.CASMotion 以 Intel 的 Core2 处理器为实验平台,利用性能监控器(performance monitor)[5]的硬件分支计数器获取 BC 值,并给出了防干扰获取时间点算法.

3.2.1 硬件辅助分支计数器简介

Intel 的 Core2 处理器系列的性能监控器(performance monitor)与以前的 P4 系列相比简化了操作,只需设置

IA32_PERFEVTSELx MSR 和 IA32_PMCx MSR 两个寄存器即可.

IA32_PERFEVTSELx MSR 称为性能事件选择寄存器(performance event select register),它占用从 186H 寄存器地址开始的一段连续的 MSR 地址空间.IA32_PMCx MSR 称为性能监控计数器(performance monitoring counter),占用从 0C1H 开始的一段连续的 MSR 地址空间.每个 IA32_PERFEVTSELx MSR 对应一个 IA32_PMCx MSR,IA32_PMCx MSR 在相应的 IA32_PERFEVTSELx MSR 控制下有选择地对事件进行计数.通过设置 IA32_PERFEVTSELx MSR 可以对相应 IA32_PMCx MSR 的事件过滤和计数行为进行控制,并且能够控制 IA32_PMCx MSR 的中断生成.

CASMotion 需要统计的是⟨*bc*,*eip*,*ecx*⟩三元组中的 BC 值,目的是统计当前虚拟机自运行以来跳转的分支数.与之相对应的是,IA32_PERFEVTSELx MSR 寄存器选择事件之一的 Branch Instruction Retired 类型事件,它具体包含 5 种事件:UNCONDITIONAL(无条件跳转)类型、CONDITIONAL(条件跳转)类型、CALL 类型、RETURN 类型和 INDIRECT 类型事件.依靠该 5 类事件可定位当前虚拟机自运行以来所在的程序分支,再考虑 EIP 和 ECX 值可最终确认指令在指令流中的位置.在泛虚拟化体系结构中,DomU 的用户态和内核态分别运行在 ring3 和 ring1 上,所以只需监控 ring1-ring3 的分支数即可.

3.2.2 防干扰获取时间点方法

由于硬件辅助的性能监控器无法识别受监控 DomU,在运行中会无差别的记数 Dom0 和各个 DomU 的程序分支.所以无论在记录阶段还是重放阶段,都需要无干扰地获取受监控 DomU 的时间点.

本文解决方法是在域间调度的间隙插入性能监控器调度函数,只记录受监控 DomU 的分支计数.在 Xen 虚拟机平台上,域间调度实际上是对 VCPU 的调度,每个 VCPU 的数据结构中已添加性能监控器的状态信息.具体操作见算法 1.由于后文需要根据 IA32_PMC0 计数触发中断,为防止其他域的干扰,当受监控 DomU 调出和调入时分别停止和重置性能计数器.

算法 1. 性能监控器调度算法.

输入:源 VCPU:*from*、目的 VCPU:*to*:

//*from*(*to*).*perfctrs.counter_enabled*:若置位表示该 VCPU 已和性能监控器建立联系;

//*from*(*to*).*perfctrs.consumed*:表示该 VCPU 自从创建以来,其运行时积累的分支计数总数;

//*from*(*to*).*perfctrs.sched_start*:表示该 VCPU 最近一次被调度时,记录下的 IA32_PMC0 的值;

步骤:

```
1.  if (to→perfctrs.counter_enabled) {        //若被调度的 VCPU 已经和性能监控器建立联系
2.      读取性能监控器信息到 VCPU 结构中;
3.      设置 PMI 中断,当计数器溢出时产生中断;
4.  }
5.  pct=read_performance_counter();           //读取 IA32_PMC0 寄存器中存储的数值
6.  if (pct 即将溢出){
7.      pct_start=1;                          //pct_start 表示被调度的 VCPU 将从该值开始记录分支计数
8.      将 pct_start 写回到 IA32_PMC0 寄存器中;
9.  } else
10.     pct_start=pct;
11. if (from 属于受监控域)
12.     停止 IA32_PMC0 计数器,但不清零;
13.     //计算源 VCPU 被调度期间的分支计数,并记录到累计值中
14.     from.perfctrs.consumed+=pct–from→perfctrs.sched_start;
15. if (to 属于受监控域)
16.     重置 IA32_PMC0 计数器,继续计数;
```

17. //保存 *pct_start*,作为该 VCPU 下次被换出时计算调度期间分支次数的依据

18. *to→perfctrs.sched_start=pct_start*;

3.3 记录方法

CASMotion 在记录阶段记录所有不确定事件发生的时间点和相关内容,以便在重放阶段精确重放.从实现记录事件的角度来分有两种实现形式:在 Xen 中插入钩子记录函数、劫持与特权域的通信.前者适用于大多数不确定事件,比如 RDTSC、VCPU 时间更新、VCPU 调度、事件通道(event channel)通知等;而后者特定应用于在前后端架构下记录设备输入信息,本研究中包括控制台设备、网络设备和 USB 设备,并且每个设备维护各自的事件记录日志.本文以记录 RDTSC 事件和控制台设备输入事件为例,分别给出记录事件的方法.

3.3.1 记录读取 TSC 时钟事件

从 Pentium 开始,所有的 Intel 80×86 CPU 都包含 64 位的时间戳记数器(TSC).该寄存器是一个不断增加的计数器,它在 CPU 启动后的每个时钟信号到来时加 1.由于 RDTSC 获取的值同物理 CPU 的启动时间相关,所以对虚拟机来说属于不确定事件.记录 RDTSC 事件的步骤如图 2 所示,Step 4~Step 9 由插入的记录事件钩子函数处理.

```
采集 RDTSC 事件步骤:
Step 1.     通过 CR4 将 RDTSC 设置为特权指令,使得 RDTSC 只能在 Xen 中被模拟执行;
Step 2.     当 RDTSC 在受监控 DomU 中执行时,触发通用保护异常(GPF);
Step 3.     在 Xen 虚拟机监控器中捕获该异常,由 do_general_protection 中断函数处理;
Step 4.     if (陷入 Xen 的 DomU 为受监控 DomU) {
Step 5.         获取 Xen 中的 struct cpu_info 结构,得到当前虚拟机的 EIP 和 ECX 值;
Step 6.         获取当前 VCPU 的运行时积累的分支计数;
Step 7.         将事件类型、时间点、tsc 时钟值封装到事件结构体;
Step 8.         将该事件结构发送至共享页队列中,若页满则通知特权域取出数据保存;
Step 9.     }
Step 10.    修改 EIP(EIP+2),使得返回后受监控 DomU 继续执行下一条指令;
```

Fig.2 Steps of recording RDTSC data

图 2 记录 RDTSC 事件步骤

RDTSC 属于数据不确定事件,在重放过程中,当受监控 DomU 发起该事件时,CASMotion 通过钩子函数截获调用并匹配时间点重放该事件.由于 RDTSC 重放过程与记录过程基本类似,本文不再累述.其他类型的不确定性事件(包括时间不确定时间),如 VCPU 时间更新、VCPU 调度、事件通道(event channel)通知等,也通过类似的步骤在各自的钩子函数中截获时间点和数据等信息,并发送到共享页队列中.

3.3.2 记录控制台设备输入事件

在 Xen 泛虚拟化的设备虚拟化结构中,虚拟机拥有前端设备而宿主机拥有后端设备,前后端设备通过事件通道和共享页传递请求和结果.事件通道端口号和共享页页号则通过 Xenstore 发布给其他虚拟机.与大多数不确定事件不同,对设备事件的记录和重放是靠劫持事件通道和共享页实现,而不是插入钩子函数.各个设备维护独自的日志文件,每条设备输入事件对应于一条事件通道通知事件.控制台设备的记录架构如图 3 所示.

Xenconsoled 是 Dom0 中处理 DomU 控制台交互的守护进程,可看作控制台后端设备.记录进程 lconsole 对 DomU 的控制台设备透明.记录进程启动后,首先截获 Xenconsoled 和 DomU 控制台设备间的通信.之后,记录进程转发 Xenconsoled 与 DomU 之间的控制台交互,并记录到相应的数据到日志文件中.当数据从 Xenconsoled 流向 DomU 时,定义该事件为非确定事件,在 lconsole 中以 RLOG_WRITE_RING 事件记录;当数据从 DomU 流向 Xenconsoled 时,定义该事件为确定性事件,在 lconsole 中触发 RLOG_READ_RING 事件,但只记录字符个数.

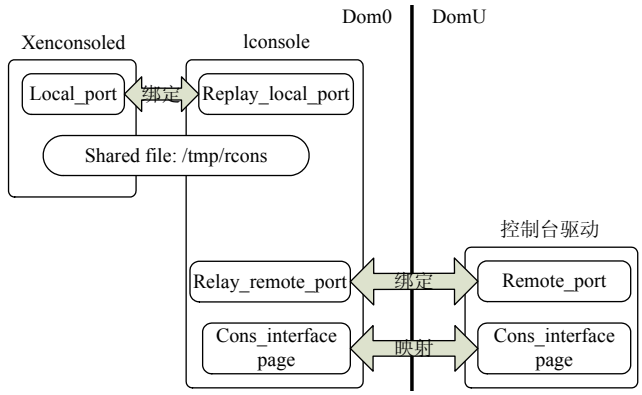

Fig.3 Architecture of recording console input event

图 3 控制台输入事件记录架构

3.4 重放方法

CASMotion 在重放阶段针对时间不确定事件、数据不确定事件和设备输入事件采取不同的重放策略.对于时间不确定事件,读出记录后需要等到事件发生的时间点才将其数据回填完成重放,所以实现的核心是时间点的等待匹配策略;而数据不确定事件的重放则相对简单,通过钩子点函数拦截事件的处理过程,若能匹配至事件发生的时间点,则回填数据完成重放,不需要计算时间点何时发生;设备输入事件的重放由事件通道(event channel)通知驱动,当特定设备的独立重放进程监控到事件通道通知事件被重放时,在共享页中填入相关日志内容完成重放.本节给出时间点的等待匹配算法,以及以控制台设备为例给出时间不确定事件(事件通道通知事件)和设备输入事件的重放步骤.

3.4.1 等待匹配时间点

当重放阶段 Xen 接收到 Dom0 传入的时间不确定事件记录时,即开启时间点等待匹配策略.该策略分为两步完成:首先,根据时间点设置性能计数器溢出中断(PMI),在接近目的分支数时触发该中断;其次,在中断处理函数中设置物理 CPU 到单步执行状态,每步匹配直至精确的时间点.最后,通过各个事件的钩子函数,回填记录中的数据,完成时间不确定事件的重放.

算法 2. 时间点等待匹配初始算法.

输入:受监控 DomU 的 VCPU:*v*,时间不确定事件的时间点:*time.bc*,*time.eip*,*time.ecx*:

//*v.perfctrs.consumed*:表示该 VCPU 自从创建以来,其运行时积累的分支计数总数;

//*v.perfctrs.sched_start*:表示该 VCPU 最近一次被调度时,记录下的 IA32_PMC0 的值;

步骤:

1. if (*v* 不为当前运行的 VCPU)
2. 初始算法失败;
3. 更新 *v.perfctrs.consumed*;
4. if (*v.perfctrs.consumed*>=*time.bc*) //分支 *time.bc* 已经发生过了
5. 初始算法失败;
6. 开启性能计数器的溢出中断,注册 PMI 中断处理函数;
7. 设置 APIC 局部向量表的 LVT 性能计数器寄存器(APIC_LVTPC),为 PMI 分配空闲的中断向量;
8. 再次更新 *v.perfctrs.consumed*;
9. *diff*=*time.bc*−*v.perfctrs.consumed*;
10. *BREAKPOINT_THRESH*=512; //阈值,当 *diff*<*BREAKPOINT_THRESH* 即触发 PMI 中断
11. if (*diff*<*BREAKPOINT_THRESH*)

12. 立即显示调用 PMI 中断处理函数,不需再等待;
13. else{
14. *t*=–(*diff*–*BREAKPOINT_THRESH*);
15. 将 *t* 回写到 IA32_PMC0 寄存器中,从 *t* 开始计数;
16. *v*.*perfctrs*.*sched_start*=*t*; //作为该 VCPU 下次被换出时计算调度期间分支次数的依据
17. }

算法 2 给出了时间点等待匹配的初始算法,第 10 步设置 *BREAKPOINT_THRESH*=512,目的是在接近目标时间点时触发 PMI 中断,再进行进一步的单步匹配,以免错过时间点.第 14 步~第 16 步中将 *t* 设置为目的分支数的负数值,当经过–*t* 个分支跳转后产生 PMI 中断.

在 PMI 中断处理函数中,只为 VCPU 设置标志寄存器 X86_EFLAGS_TF,进入单步执行状态.随后,该 VCPU 每执行一个指令都会陷入 Xen 中,并试探匹配时间点.具体匹配时间点的过程见算法 3.

算法 3. 精确匹配时间点算法.

输入:目标时间点:*time*.*bc*,*time*.*eip*,*time*.*ecx*;

步骤:

1. if (陷入的 Dom 不为受监控 DomU ||当前 VCPU 不运行在单步调试状态)
2. 不予匹配,退出;
3. 取得当前受监控 DomU 的寄存器 *regs*.*eip* 与 *regs*.*ecx*;
4. *r*=*get_pcounter_val*(); //*r* 为当前受监控 DomU 的运行以来执行过的分支数
5. while ((*time*.*eip*!=0 && *regs*.*eip*!=*time*.*eip* && *r*==*time*.*bc*) ||*r*<*time*.*bc*) {
6. if (*regs*.*eip* 指向的指令为 int,*iret*,*popf* 或者 *pushf*) {
7. 则模拟执行该指令;
8. 更新 *regs*.*eip*,指向下一条指令;
9. }
10. else
11. 不予匹配,退出;
12. }
13. if (*time*.*eip*!=0 && *regs*.*ecx*!=*time*.*ecx* && *r*<=*time*.*bc*) {
14. if (*regs*.*eip* 指向的指令具有 REP 或 PEPNE 前缀) { //指令需循环执行,并由 *ecx* 控制剩余次数
15. while (*regs*.*ecx* > *time*.*ecx*) {
16. 模拟执行该指令,执行完毕后不更新 *regs*.*eip*;
17. *regs*.*ecx*––;
18. }
19. }
20. else
21. 不予匹配,退出;
22. }
23. if ((*r*==*time*.*bc*) && (*regs*.*eip*==*time*.*eip*) && (*regs*.*ecx*==*time*.*ecx*)){ //匹配成功
24. 插入时间不确定事件,通知受监控 DomU 处理;
25. 清除已重放的事件,并读入下一条预备重放的事件不确定事件;
26. 取出时间不确定事件的时间点:*time*.*bc*,*time*.*eip*,*time*.*ecx*;
27. *regs*.*eflags* &=~X86_EFLAGS_TF; //关闭 VCPU 的单步运行状态
28. 跳转至“算法 2”;
29. }
30. else
31. 退出;

算法 3 的第 5 步~第 12 步判断是否已执行到目的分支数.由于本研究透明地重放虚拟机执行过程,所以需要避免使得受监控 DomU 感知已设置 TF 位,并且某些异常处理也会清除 TF 位.所以第 6 步~第 9 步模拟执行有可能读取或更改 VCPU 设置标志寄存器的指令,包括 *int*,*iret*,*popf* 和 *pushf* 这 4 种.第 13 步~第 22 步的含义是,如果分支数和指令指针都已匹配,并且该指令具有 REP 或 PEPNE 前缀即需循环执行,则模拟该指令直至匹配 ecx 寄存器.若算法 3 中途执行失败退出,则受监控 DomU 会在执行下一条指令时继续陷入 Xen 并试图匹配时间点.

第 24 步是插入时间不确定事件,每个事件的处理过程各不相同.本文以事件通道通知事件为例,给出插入该事件的步骤.在重放阶段插入事件通道(event channel)通知事件的步骤如图 4 所示.

```
插入事件通道通知事件步骤:
输入:受监控 DomU 的 VCPU:v,事件通知端口号:port,事件记录时的状态标识:past_flag;
Step 1.        s=v→domain→shared_info;   //获得受监控 DomU 的共享信息页
Step 2.    cur_flags=0;
Step 3.        if (s→pending 未被置位) {
Step 4.            置位 s→pending;
Step 5.            set_bit(1, & cur_flags);
Step 6.            if (s→evtchn_mask 未被置位) {
Step 7.                set_bit(2, & cur_flags);
Step 8.                if (evtchn_pending_sel 中未置位 port 相关位置) {
Step 9.                    置位 evtchn_pending_sel;
Step 10.               set_bit(3, & cur_flags);
Step 11.               设置 upcall,发送事件;
Step 12.           }
Step 13.       }
Step 14.   }
Step 15.   if (past_flags!=cur_flags)
Step 16.       报错,停止重放过程;
```

Fig.4 Steps of replaying event channel notify event

图 4 插入事件通道通知事件步骤

3.4.2 重放控制台设备输入事件

控制台设备的重放是由独立的重放进程完成,其拦截控制台输入的原理与记录阶段相同.每当插入事件通道通知事件发生时,重放进程根据日志记录决定是否回填控制台输入以完成设备输入重放.具体步骤如图 5 所示,如果收到 *replay_remote_port* 通知,如 Step 5,则说明 CASMotion 已经成功重放了与该条记录相关的事件通道事件,控制台重放进程则回显受监控 DomU 处理结果并填入下一条事件记录.

```
重放控制台设备输入事件步骤:
输入: 受监控 DomU 的 ID:remote_dom
Step 1.        建立初始化连接;
Step 2.        读入下一条需要重放的控制台设备输入记录;
Step 3.        将记录放入 remote_interface_in 中,受监控 DomU 可在事件通道事件发生时读取该记录;
Step 4.        事件发生,读出目的端的端口号 port;
Step 5.        if (port==replay_remote_port) {   //CASMotion 已重放事件通知事件,受监控 DomU 将处理结果显示到控制台
Step 6.            while (remote_interface_out 中有数据) {
Step 7.                从 remote_interface_out 读出字符串,转写到 local_interface_in 中;
Step 8.                更新 remote_interface_out 页环;
Step 9.            }
Step 10.       通知端口 replay_local_port;      //Xenconsoled 显示执行结果
Step 11.       跳转至 Step 2.
Step 13.   }
Step 14.   else
Step 15.       跳转至 Step 4.
```

Fig.5 Steps of replaying console input event

图 5 重放控制台设备输入事件步骤

4 实验评估

CASMotion 基于 Xen3.3 和 Intel服务器平台实现.Dom0 控制域使用 32 位 Fedora8 操作系统,Linux 内核版本为 2.6.18.受监控 DomU 为泛虚拟化架构下单 VCPU 的 32 位 Fedora 9 操作系统,Linux 版本为 2.6.18,使用内存 2GB.Intel 服务器平台采用 4 核 Core2 Quad Q9400 2.66G 处理器和 4GB DDR2 内存.

为评估重放系统对 DomU 运行性能的影响,分别在相同的未监控 DomU、记录时 DomU 和重放时 DomU 上运行测试程序,并比较测试程序的运行时间.测试程序为 SPEC CPU2006 中的 400.perlbench,401.bzip2,403.gcc,429.mcf,445.gobmk,456.hmmer,464.h264ref,471.omnetpp,473.astar,483.xalancbmk,测试数据集为相应的 base 数据集.测试结果如图 6 所示,在记录阶段,受监控 DomU 运行性能的损耗在 3%之内,其中,损耗在 2%之上的为 403.gcc 和 429.mcf.在重放阶段,平均性能损失提升至 8%左右.但是,SPEC CPU2006 中的测试程序并不直接和不确定性事件相关,所以该测试的目的是评估不同程序的指令在执行时出现延迟的情况,借此说明重放系统对 DomU 中运行程序的普遍影响.

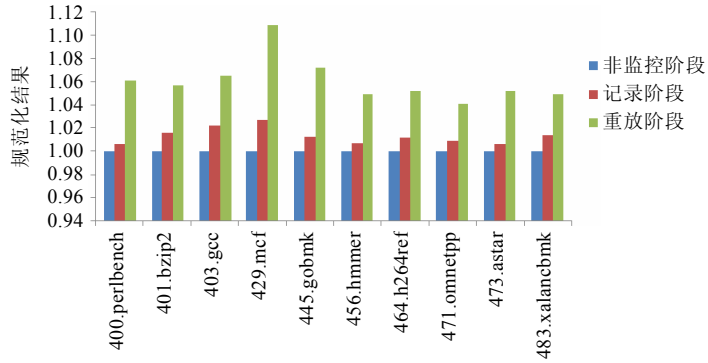

Fig.6 Performance test of monitored DomU

图 6 受监控 DomU 性能测试

本文使用 UnixBench 工具评估了重放系统对 Dom0 控制域的影响,即分别测试运行未监控 DomU、记录时 DomU 和重放时 DomU 期间 Dom0 控制域的负载.取单并发进程的测试结果,图 7 为规范化的比较结果.当 Dom0 开始记录 DomU 的状态时,Dom0 中运行程序的平均性能下降在 8%之内.而在重放 DomU 阶段,Dom0 的性能损失也不会超过 5%.所以,重放系统在 Dom0 中的控制程序不会对 Dom0 本身产生大量性能损失,且记录阶段和重放阶段对 Dom0 的影响差别也在 2%之内.

CASMotion 日志文件的增长情况见表 1.依照日志文件增长率,一个 60G 的磁盘可以容纳大约 57 周(13 个月)的日志数据,这样的日志增长率能够适用于大部分实际应用.其中,控制台输入的数据量与用户的交互输入直接相关,事件通道事件频率与用户击键频率成正比.

本文使用 Xentrace 测试工具在 Dom0 中对 DomU 中的控制台端口的访问情况进行统计.图 8 给出连续 10 次测试的控制台端口请求的平均处理时间,该时间表示为相关操作的 TSC 周期数.由于在 Dom0 中对 TSC 周期数进行测试,所以重放系统对 DomU 的 rdtsc 指令的重放过程不会干扰测试结果.由测试结果可得,记录阶段 DomU 的控制台端口请求操作会额外付出 50%左右的 TSC 周期数.而重放阶段 DomU 则会付出 4 倍左右的额外 TSC 周期数.所以,重放系统对不确定性事件的重放会导致巨大的性能开销.但由于不确定性事件指令在单核虚拟机的指令序列中所占比例较小,重放系统的整体开销仍可以控制在可接受范围之内.

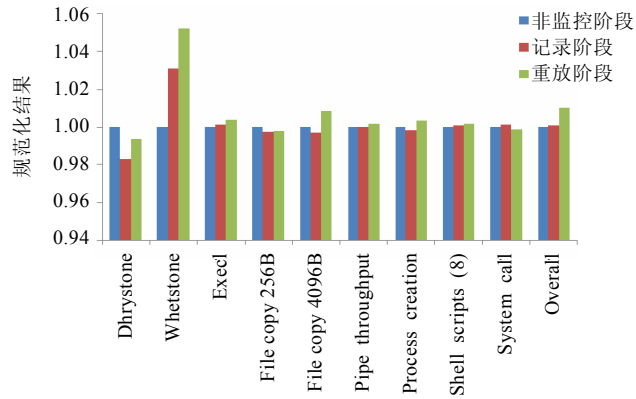

Fig.7 Performance test of host operating system

图 7 宿主操作系统性能测试

Table 1 Size of record log

表 1 日志体量

更新事件	RDTSC	控制台输入	其他	总计
0.077G/天	0.069G/天	用户输入相关	0.002G/天	约 0.150G/天

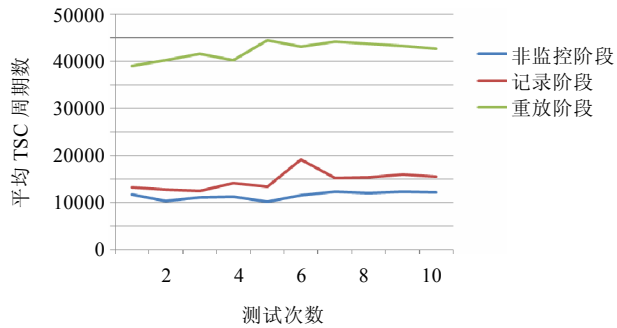

Fig.8 Performance test of console access

图 8 控制台端口的访问效率测试

5 相关工作比较

应用执行重放技术重现不确定错误,最初是由研究进程重放开始的,近几十年得到了较大发展.但进程重放技术限于实现过于高层(常在运行库、应用层实现)以及在表达系统状态方面的不足,近年来,基于虚拟机、模拟器的重放技术逐渐兴起.

目前,影响较大的基于虚拟机的系统重放工作主要有 VMWare 研发的重放器、ExecRecorder 和 Revirt 系统.应用领域涵盖了系统调试、虚拟机迁移和入侵检测等方向.但就应用于系统调试领域的虚拟机重放系统来说,目前还未能从模型分析的角度给出适用于调试场景的重放过程形式化描述方法,进而无法说明完成的系统是否具有准确重放虚拟机系统内部状态的能力.

VMWare 重放器[6]基于 VMWare Workstation 开发,在虚拟机管理器中记录和重放不确定事件.通过对记录的裁剪,可对指定的 perl 脚本程序、进程或者全系统进行重放,目前只支持单核客户虚拟机.但由于 VMWare Workstation 虚拟机管理器运行在宿主操作系统之上,通过软件模拟客户操作系统的硬件,效率较低也不是主流

的虚拟机管理器架构,所以其应用前景受到限制.

ExecRecorder[7]是加利福尼亚大学于2006年研发的一个系统,它构建在Bochs模拟器上,实现了全系统级别的执行过程日志记录、运行重放与故障恢复功能.通过 fork 系统调用对当前的虚拟机进程的状态进行备份,在虚拟机进程中存放了 Bochs 中的 CPU、内存、时钟与中断、外设这 4 个系统状态.ExecRecorder 开销较大,每产生一次检查点就需要创建新的进程,极大地消耗了内存空间.另外,由于 ExecRecorder 基于 Bochs 实现,然而 Bochs 本身只能作为模拟器来使用,主要用来实现对硬件进行软件模拟,速度很低,这也是 Bochs 的一个缺陷.

Revirt[8]是一款在 UMLinux(user mode Linux)虚拟机监控器上实现的基于单处理器的虚拟机全系统执行重放系统.在 Revirt 中记录的不确定性事件包括异步中断、外部输入以及运行在用户态下时会产生不确定性结果但又不会正常陷入的指令.在重放阶段,Revirt 禁止产生新的异步中断以免影响系统当前的重放进程.当重构原始的异步中断时,同样使用硬件计数器与系统钩子相结合的方法.后续工作 SMP-Revirt[9,10]又在 Xen 和 UMLinux 系统上应用 CREW 协议,实现了对多处理器系统中共享对象状态的执行重放.

本文的工作与 SMP-Revirt 同样基于 Xen 虚拟机系统,也借鉴了 Revirt 系统的设计思路,但两者之间有诸多区别:其一,SMP-Revirt 是密歇根州立大学为其入侵检测系统开发的重放工具[14],目的是在系统底层记录受攻击的全过程,并在稍后重现攻击过程.本文是针对虚拟机系统调试而研发的重放工具,要求在重放阶段系统状态与记录阶段一致.本文首先引入虚拟机指令状态表达模型,提出虚拟机重放定义并证明了完成系统重放的充分条件.根据该充分条件并结合 Xen 虚拟机系统讨论不确定事件范畴,最终实现对单核虚拟机系统的执行重放;其二,在具体实现技术上本文也有改进.本文使用的性能计数器基于 Intel Core2 处理器平台而非 Revirt 的 Intel P4 平台,操作更加简化、适用范围更加广泛.此外,相对研发 SMP-Revirt 时的 Xen 系统,目前新版的 Xen 虚拟机系统大量使用了 Xenstore 等域间共享技术,因此,CASMotion 重新设计实现了诸如设备重放进程、事件获取等模块.

与本文相似的工作主要有华中科技大学的 XLS 和浙江大学的 Bbreplayer.XLS[11]是基于 Xen 的不确定事件记录系统,它属于性能计数器完成对键盘事件和虚拟机时间更新事件的记录.本文与它的区别主要有 3 部分:(1) XLS 缺乏对记录不确定事件应用能力的准确描述,无法确定 XLS 将会被正确地应用于调试、入侵检测等领域;(2) XLS 继续适用 Intel P4 提供的性能计数器,无法兼容 Intel Core2 以及后续的 Intel i 系列处理器,不能应用于主流处理器体系架构,而本文解决了该问题;(3) 在新版 Xen 虚拟机系统上虚拟机时间被分化为 VCPU 时间和墙上时间两部分,CASMotion 分别对这两种时间分别记录、重放.Bbreplayer[12]是基于 Bochs 模拟器开发的重放系统,效率比较低.而且,它的不确定事件的时间点依赖于 CPU 的执行过的时间片,难以论证其重放时对操作系统内部状态一致性的保持.

6 总结与未来工作

基于虚拟机的系统执行重放技术为客户机操作系统提供高可靠性保证,特别在虚拟机调试、虚拟机迁移以及虚拟机入侵检测等方向具有非常广阔的应用前景.本文为使得执行重放系统适用于系统调试,建立虚拟机指令执行模型,提出了虚拟机执行重放的定义,并用代数表达式形式化地证明成功重放的充分条件.根据该充分条件设计实现了基于 Xen 的虚拟机重放系统 CASMotion.CASMotion 讨论了 Xen DomU 中不确定事件的种类,给出各类事件的重放方法以及时间点的匹配算法.下一步继续将 CASMotion 拓展到多 VCPU 环境下,可完成对系统并发错误的重现.

References:

[1] Cornelis F, Georges A, Christiaens M, Ronsse M, Ghesquiere T, De Bosschere K. A taxonomy of execution replay systems. In: Proc. of the Int'l Conf. on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet. 2003.

[2] Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A. Xen and the art of virtualization. In: Proc. of the 19th ACM Symp. on Operating Systems Principles. Bolton: ACM Press, 2003. 164–177. [doi: 10.1145/945445.945462]

[3] Popek GJ, Goldberg RP. Formal requirements for virtualizable third generation architectures. Communications of the ACM, 1974, 17(7):412−421. [doi: 10.1145/361011.361073]

[4] Damani OP, Garg VK. How to recover efficiently and asynchronously when optimism fails. In: Proc. of the 16th Int'l Conf. on Distributed Computing. 1996. 108−115. [doi: 10.1109/ICDCS.1996.507907]

[5] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B System Programming Guide. Part 2. 2007.

[6] Chow J, Lucchetti D, Garfinkel T, Lefebvre G, Gardner R, Mason J, Small S, Chen PM. Multi-Stage replay with crosscut. In: Proc. of the 6th ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual Execution Environments (VEE 2010). New York: ACM Press, 2010. 13−24. [doi: 10.1145/1735997.1736002]

[7] Oliveira DAS, de Crandall JR, *et al*. ExecRecorder: VM-based full-system replay for attack analysis and system recovery. In: Proc. of the 1st Workshop on Architectural and System Support for Improving Software Dependability. San Jose: ACM Press, 2006. 66−71. [doi: 10.1145/1181309.1181320]

[8] Dunlap GW, King ST, *et al*. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. SIGOPS Operating System Review, 2002,36(SI):211−224. [doi: 10.1145/1060289.1060309]

[9] Dunlap GW. Execution replay for intrusion analysis [Ph.D. Thesis]. University of Michigan, 2006.

[10] Dunlap GW, Lucchetti DG, *et al*. Execution replay of multiprocessor virtual machines. In: Proc. of the 4th ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual Execution Environments. Seattle: ACM Press, 2008. 121−130. [doi: 10.1145/1346256.1346273]

[11] Pan ZQ. A non-deterministic events log system based on the paravirtual VM Xen [MS. Thesis]. Wuhan: Huazhong University of Science and Technology, 2008 (in Chinese with English abstract).

[12] Ma C. Research on VM-based incremental checkpoint and execution replay [MS. Thesis]. Hangzhou: Zhejiang University, 2008 (in Chinese with English abstract).

附中文参考文献:

[11] 潘正秋.基于半虚拟化 Xen 的非确定性事件记录系统[硕士学位论文].武汉:华中科技大学,2008.

[12] 马超.基于虚拟机的增量检查点和执行重放技术研究[硕士学位论文].杭州:浙江大学,2008.

于佳耕(1983－),男,山东济南人,博士生,主要研究领域为虚拟机技术.

武延军(1979－),男,博士,副研究员,主要研究领域为操作系统,系统安全.

周鹏(1984－),男,工程师,主要研究领域为系统安全,虚拟机技术.

赵琛(1967－),男,博士,研究员,博士生导师,主要研究领域为编译技术.