基于微内核的虚拟机 I/O 安全机制①

王玉菁[1,2], 吴 涛[1,2], 杨秋松[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](中国科学院大学, 北京 100190)

摘 要: NOVA 等微内核虚拟化架构解决了宏内核平台可信计算基体积和攻击面过大的问题, 但其仍缺乏虚拟机分等级保护和 I/O 资源访问控制等安全机制. 本文提出了安全域的概念, 并将虚拟机划分至不同的安全域, 进而建立可定制的 I/O 资源访问控制机制. 通过将访问控制模块添加至 I/O 资源访问的关键代码路径, 实现了不同安全域的 I/O 资源访问控制. 实验表明, 该机制提高了数据的隔离性与安全性, 仅对计算密集型、I/O 密集型任务造成了较小的性能损耗.

关键词: 微内核; 虚拟化; 安全域; I/O 资源访问控制

I/O Security Mechanism for Microhypervisor Based Virtualization Architecture

WANG Yu-Jing[1,2], WU Tao[1,2], YANG Qiu-Song[1]

[1](Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Science, Beijing 100190, China)

Abstract: Micro-kernel based virtualization architectures such as NOVA solves the problems of large trusted computing base and attack surface in most macro-kernel based virtualization systems. However, NOVA lacks of protections for the virtual machines of different security levels separately. Also, it lacks of access control mechanisms of the virtual machines to the I/O resources. In this paper, we propose the concept of security regions and introduce a way to divide virtual machines into several security regions, upon which the I/O resources access control mechanism is built. To implement I/O resource access control between different security regions, this mechanism adds an access control module to the key code path between the virtual machine monitor to the I/O services. The experiments show that when promoting the isolation and safety of data, it only impacts the performance of CPU bound tasks and I/O bound tasks slightly.

Key words: microhypervisor; virtualization; security level region; I/O access control

1 引言

虚拟化技术是计算领域的一项传统技术, 它允许多个具有不同操作系统的虚拟机独立运行在一个物理机上, 实现了底层资源的隔离和有效使用. 目前虚拟化技术在业界有着广泛的应用, 如使用 Xen 虚拟化技术的亚马逊 EC2(Elastic Cloud Computing)云计算平台等[1].

然而虚拟化技术也引入了一系列风险. 大多数虚拟化平台基于宏内核. 以 Xen 为例, 其可信计算基 TCB(Trusted Computing Base)包括系统管理程序(Hypervisor)和对普通虚拟机进行管理的 Domain 0. 由于系统管理程序和 Domain 0 均运行于 0 环, 这两部分代码量庞大, 攻击面相应较大, 并且由于所有虚拟机都需要通过 Domain 0 来管理和获取资源, 一旦 Domain 0 受到攻击, 整个虚拟机系统将受到影响, 安全性难以保证[2].

针对以上安全性问题, 德累斯顿工业大学设计并开发了以 Intel–VT 或 AMD–V 为基础、基于微内核的新型虚拟化架构 NOVA[3]. NOVA 的 TCB 包括微型系统管理程序(Microhypervisor, 9 KLOC), 精简的用户

① 基金项目:国家自然科学基金(61303163,91218302,61432001);中国科学院重大方向性项目(KGCX2–YW–125)
收稿时间:2015-01-22;收到修改稿时间:2015-03-18

级环境(27 KLOC). 相较于传统虚拟化架构的 TCB 规模, 如 Xen(300 KLOC), VMware ESXi(200 KLOC), NOVA 解决了 TCB 过大带来的一些问题, 系统的安全可靠性得到相应提升[3]. 然而 NOVA 作为一种较新的虚拟化架构, 其在 I/O 安全方面仍然存在一定问题:

首先, 在同一个系统中提供不同安全等级的虚拟机隔离机制是十分必要的, 是用户数据安全保障的关键. 但是, 目前 NOVA 对虚拟机的安全等级没有区分, 并且无法针对不同安全等级的虚拟机可访问的 I/O 资源进行控制, 无法确保系统中的关键 I/O 资源被合理使用, 因此存在潜在的数据安全隐患. 例如, 系统中某些虚拟机需要使用 USB 外部设备进行身份认证(USB Token), 而这些 Token 由于密级较高而不便被其他虚拟机访问. 为了保证其完整性和机密性, 系统需要提供相应的访问控制机制限制该 Token 不被其他安全等级较低的虚拟机访问. 另外, 对于安全等级较高的虚拟机, 由于其存储和处理的数据具有较高的机密性, 往往需要禁止其访问网络服务, 进而消除这些高机密性数据发生泄漏的潜在威胁.

其次, NOVA 系统由于缺少虚拟机 I/O 资源的访问控制机制, 导致其对 I/O 资源的共享方式也无法加以限制. 安全的虚拟化系统应当控制 I/O 资源在不同虚拟机间的共享权限. 譬如: 具有竞争关系的 A 公司与 B 公司各租用了同一系统内的若干台虚拟机, A 公司出于对数据安全性的考虑, 希望独享磁盘块 d, 那么系统应当通过访问控制机制, 限制该磁盘块 d 仅在 A 公司租用的各虚拟机间共享, 而禁止 B 公司租用的虚拟机访问该磁盘块 d.

针对上述安全问题, 本文将在 NOVA 虚拟化架构下, 提出一种新的面向 I/O 资源访问控制的安全机制. 接下来, 第 2 节将对该安全机制涉及的访问控制相关的概念及技术进行阐述, 第 3 节将详细介绍该安全机制的设计方案, 第 4 节针对该安全机制对系统整体性能产生的影响进行实验与结果分析, 最后是总结.

2 访问控制

2.1 访问控制概念

访问控制是信息安全保障机制的核心内容, 它是实现数据保密性和完整性机制的主要手段. 访问控制通过某种途径显式准许或限制访问主体(发起者, 是一个主动的实体)对访问客体(需要保护的资源)的访问权限, 从而使计算机系统在合法范围内使用, 限制对关键资源的访问.

2.2 访问控制实现技术

访问控制矩阵是一套基本的访问控制抽象模型, 于 1971 年由 B.W.Lampson 提出. 其以一种形式化的方法对访问控制进行抽象, 利用二维矩阵规定了任意主体到任意客体间的访问权限. 主体是访问操作中的主动实体, 客体是访问操作中的被动实体. 矩阵中每一格代表所在行的主题是否对所在列的客体具有访问权限.

由于在实际系统应用中, 访问控制矩阵往往较大, 很多格可能为空, 为了避免保存这样稀疏的矩阵造成的空间浪费, 因此常用的方式有两种:

① 以访问控制矩阵的列划分, 为每一列建立一个访问控制列表. 例如 Linux 操作系统就以文件为中心建立了访问控制列表, 设置特定的用户或用户组对一个文件或目录的操作权限[4].

② 以访问控制矩阵的行划分, 为每一行建立一个访问权限表[5].

2.3 传统虚拟机系统中的访问控制

以传统虚拟机系统 Xen 为例, 其使用 sHype 提供访问控制和虚拟资源的隔离等服务[6]. Xen 的访问控制模块 ACM 允许 Domain 0 的用户自定义安全策略, 当虚拟机请求访问其他虚拟机或资源时, ACM 模块将根据用户制定的安全策略以及虚拟机的安全标签进行判断, 继而达到针对虚拟机可访问的资源进行控制的目的.

3 I/O安全机制的设计

本文基于 NOVA 虚拟化架构, 针对其现存的 I/O 安全问题, 提出了一种新的 I/O 安全机制. 为了阐明安全机制的设计, 本章首先对 NOVA 虚拟化架构进行简要介绍.

NOVA 将虚拟化层分解为虚拟机监控器(Virtual Machine Monitor)、微型系统管理程序和根域管理器(Root Partition Manager), 如图 1 所示. 其中仅有基于微内核的微型系统管理程序运行在内核空间, 提供包括线程调度、地址空间管理以及进程间通信机制等在内的一小部分硬件的抽象, 其余包括根域管理器、虚拟机监控器以及用户操作系统均运行于用户态. 在传统虚拟化架构中, 虚拟机监控器等同于系统管理程

序、直接运行在硬件之上处于特权模式、负责隔离和管理上层运行的多个虚拟机, 而在 NOVA 中, 虚拟机管理器运行于用户态, 每个虚拟机管理器对应一个虚拟机, 负责管理虚拟机和主机系统的物理资源之间的交互并模拟出需要的硬件环境. NOVA 系统的大部分功能由运行在用户态的服务程序完成.

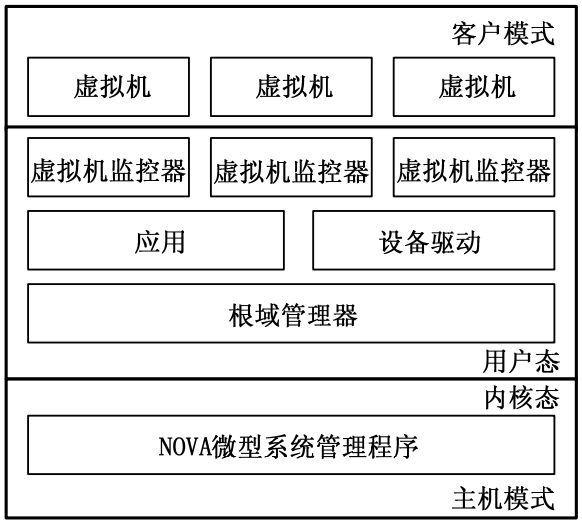

图 1　NOVA 微内核虚拟化架构

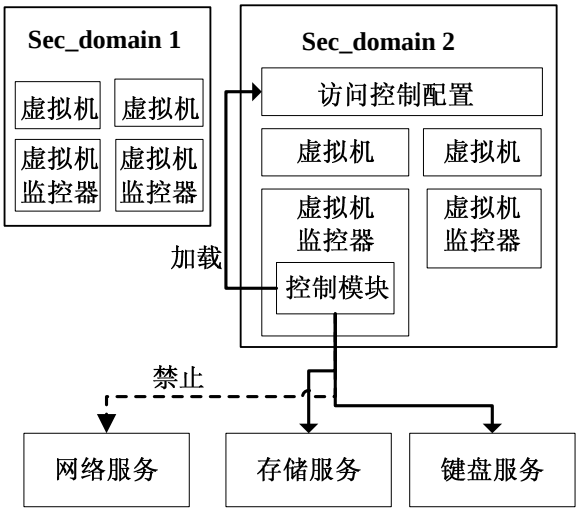

图 2　I/O 安全机制设计方案

根据上述 NOVA 虚拟化架构介绍以及引言中指出的 NOVA 现存 I/O 安全问题, 本文提出了一种新的 I/O 安全机制, 如图 2 所示. 首先为虚拟机引入安全等级标签, 根据安全等级标签将虚拟机划入不同的安全域; 支持 I/O 资源访问控制策略的动态配置, 对属于不同安全域的虚拟机分别采取相应的针对 I/O 资源的访问控制配置; 在虚拟机监控器中加入访问控制模块, 该模块载入虚拟机所在的安全域的访问控制配置, 根据配置允许或禁止虚拟机访问相应的 I/O 资源.

3.1 安全等级标签与安全域划分

在同一个虚拟化系统中, 往往需要运行多个处于不同安全等级的虚拟机. 如何对于这样的大型系统内的虚拟机进行分等级的保护, 而非进行同一等级的保护成为了一个重要的安全问题. 为此, 我们在 NOVA 系统中引入了安全域的概念.

安全域是指在同一系统中, 具有相同或相似的安全保护需求和保护策略, 并相互信任的元素的集合. 安全域往往根据安全等级进行划分. 因此在 NOVA 系统中, 为了对虚拟机进行安全域的划分, 首先我们需要指定虚拟机的安全等级 Se. 假定{DOM_1, DOM_2, DOM_3, …, DOM_n}是当前系统虚拟机集合{VM_1, VM_2, VM_3, …, VM_m}基于安全域的一个划分, 那么满足:

对于任意 *VMi*,*VMj*, 若 *VMi*,*VMj* 属于同一个安全域 *DOMp*, 当且仅当 *Se*(*VMi*)=*Se*(*VMj*).

为了表示的方便与直观, 我们采用整数表示虚拟机的安全等级. 由于 NOVA 中每个虚拟机监控器上只运行一个虚拟机, 因此我们可以将虚拟机的安全等级标签加在其对应的虚拟机监控器实例中. 虚拟机监控器在 NOVA 的运行时环境 NRE(NOVA Running Environment)中对应 Vancouver 类. 通过给 Vancouver 类添加表示安全等级的 security_level 属性, 系统在创建虚拟机时, 为相应的 Vancouver 实例赋予指定的安全等级标签.

当系统中的虚拟机(包括其对应的虚拟机监控器)被打上安全标签后, 安全标签相同的虚拟机自动被归为同一个安全域. 由于同一安全域内的虚拟机对于安全保护有相同的需求, 系统对于同一安全域的虚拟机执行相同的 I/O 资源访问控制. 具体将在 3.2 与 3.3 节中分别讨论.

3.2 访问控制矩阵定义

根据 2.2 节中访问控制矩阵的介绍, 这里, 我们同样采用矩阵的方式表示不同安全域的虚拟机可访问的 I/O 资源. 矩阵的列分别代表不同的 I/O 资源, 如网卡、键盘和硬盘(可细化至各硬盘块)等. 矩阵的行代表不同的安全域. 矩阵的值为 1 和 0, 分别表示该安全域对对应的 I/O 资源是否有访问权限.

表 1 即为一个简单的安全域—I/O 资源访问控制

矩阵. 其中安全域 Sec_domain1 可访问的 I/O 资源包括键盘以及硬盘块 2; 安全域 Sec_domain2 可访问的 I/O 资源包括网卡、键盘以及硬盘块 1.

表 1 安全域—I/O 资源访问控制矩阵示例

	Network Card	keyboard	hard disk drive1	hard disk drive2	…
Sec_ domain1	0	1	0	1	
Sec_ domain2	1	1	1	0	
…					

我们将采用访问权限表的方式来描述每个安全域的虚拟机可访问的 I/O 资源. 以表 1 中的 Sec_domain1 安全域为例, 其访问权限表为:

Sec_domain1: keyboard, hard disk drive2

3.3 I/O 资源访问控制定制

在引入了安全域概念和访问控制矩阵的基础上, 需要为不同的安全域分别进行相应的 I/O 资源访问控制. 为了完成这一目标, 还需要跟踪虚拟机通过虚拟机监控器进一步和 NRE 中抽象 I/O 设备模块进行交互的代码路径, 并根据访问权限表的设置判断是否需要切断数据的交换. 在本节中, 首先分析 NRE 中虚拟机对 I/O 资源的访问方式, 之后将以具体的例子说明访问控制的实现方式.

NOVA 作为一个基于微内核的虚拟化系统, 相比较宏内核, 其内核中大部分服务功能模块以及所有驱动程序都被移动到了内核之外, 即在用户态实现. 当有进程需要请求该服务时, 该进程通过系统调用接口向内核提出服务申请, 系统调用接口在收到该申请后, 向处于用户态的服务进程发送消息, 启动该服务进程, 并向发起服务请求的进程提供相应的服务. 一般将请求服务的进程记做“客户端”, 将提供服务的进程记做“服务器端”, 这种通信模式称为客户端–服务器通信方式[7].

NOVA 系统中, 虚拟机对于 I/O 资源的访问就是基于客户–服务器的通信方式. 客户端通过两种方式与服务器进行通信:

①对于需要进行数据块传递的服务, 例如磁盘读写、网络访问等, NOVA 系统通过共用地址空间结合生产者—消费者同步模型实现客户端与服务器之间数据包的传递. 以客户端向服务器端传输数据为例, 如图 3 所示, 客户端会话作为数据的发送方, 即生产者, 将数据写入与服务器端的消费者所共用的地址空间中的环形缓冲器, 并对与消费者所共享的信号量执行 up 操作, 该信号量处于阻塞状态直至服务器会话的消费者执行 down 操作, 并取出数据后进行相应的处理.

②通过系统调用将用户线程控制块中的数据发送至服务器端. 通过用户线程控制块, 客户端可以向服务器端发送一些命令式请求. 例如客户端可以将请求网卡物理地址的命令写入用户线程控制块的数据区, 执行相应的系统调用, 服务器端在收到用户线程控制块后解析该命令并调用驱动中获得网卡物理地址的相关函数, 将返回值写回至该用户线程控制块、设置相应标志位并发回至客户端. 此外, 用户线程控制块还可用于地址空间、信号量等内核对象权能的传递、授予与撤销等.

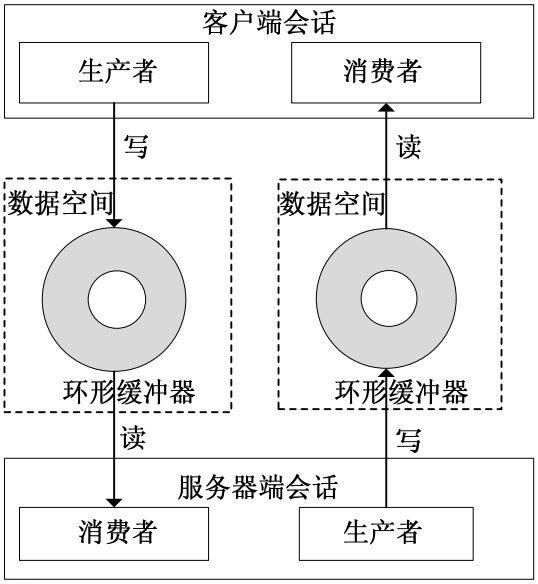

图 3 客户端—服务器数据块传递方式

以网络服务为例, 如图 4 所示, 在 NRE 中包含网络服务的模块, 即通信的服务器端. 该模块位于用户态环境下, 与系统网卡驱动程序相关联. 在 NRE 系统初始化时, 它向负责管理 I/O 端口、数据空间和服务的服务管理器(Child Manager)注册网络服务, 并启动该网络服务. 相应的, 通信的客户端位于虚拟机监控器中. 当新的虚拟机被创建时, 该虚拟机对应的虚拟机监控器, 即 Vancouver 类的实例被初始化, 该实例中创建了一个全局线程, 该线程利用客户端会话与网络服务进行通信. 当虚拟机需要访问网络 I/O 资源时, 对应的虚拟设备总线接收到相关消息信号, 并进一步交

由与网络服务通信的客户端会话. 客户端会话在解析该消息后, 采用上述分析中提到的两种客户端—服务器通信方式之一将消息发送至网络服务模块, 最终消息传入系统的网卡驱动程序进行处理.

根据上述虚拟机对 I/O 资源访问方式的分析, 我们通过控制位于虚拟机监控器中、与特定的 I/O 资源服务相关联的客户端服务会话数据的发送, 来完成相应的访问控制.

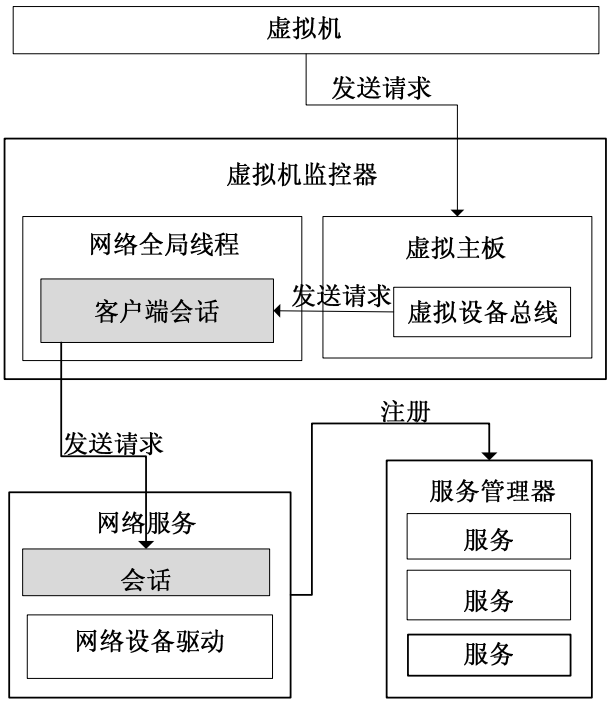

图 4　虚拟机访问网络服务流程跟踪

结合 3.1 节与 3.2 节中的分析, 我们首先需要在各虚拟机的配置文件中加入安全等级配置, 同时将安全域—I/O 资源访问控制矩阵以访问权限表的形式写入配置文件, 在 NOVA 系统初始化时载入该配置文件.

当虚拟机被创建时, 系统读取该虚拟机的安全标签, 写入其对应的 Vancouver 中的安全等级属性, 并根据配置文件解析出该虚拟机对应的安全域的访问权限表, 存入 Vancouver 中访问权限表对应的数据结构.

为了实现访问控制, 根据上述虚拟机通过虚拟机监控器进一步和 I/O 设备抽象模块进行交互的代码路径的分析, 我们在 Vancouver 类中添加了一个静态访问控制模块, 该模块截获所有虚拟机向虚拟机监控器发来的访问 I/O 资源的请求消息, 并与访问权限表进行对比, 判断当前的虚拟机是否有权通过客户端会发将请求消息进一步发送至服务器端会话并完成相应的 I/O 资源访问.

结合表 1 的访问控制矩阵, 还是以网络 I/O 资源的访问控制为例, 由于 Sec_domain2 具有访问 Network card 的权限而 Sec_domain1 禁止访问, 结合图 4, 我们通过截获虚拟机监控器接收到的虚拟机请求访问网络资源的消息, 进行访问规则的比对, 禁止属于 Sec_domain1 的虚拟机监控器通过客户端会话向提供 I/O 资源服务的服务器端会话传输请求数据, 从而控制了本安全域内的虚拟机对网络 I/O 资源的访问, 如图 5 所示.

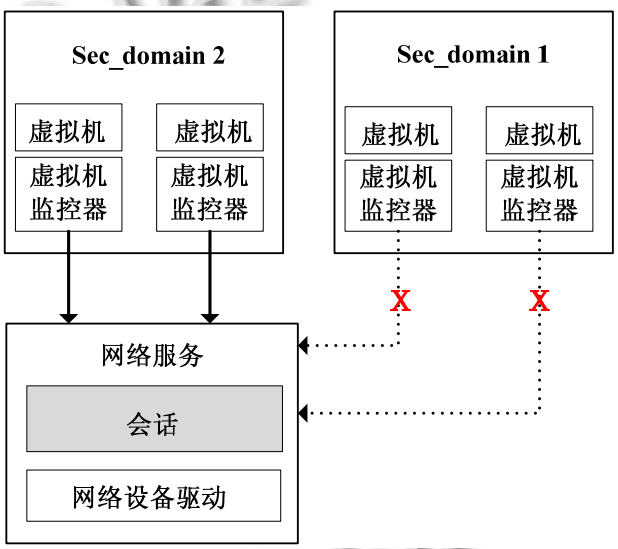

图 5　安全域—网络 I/O 资源访问控制示例

4　实验与结果分析

本节主要针对 I/O 访问控制模块对系统整体性能产生的影响进行分析.

4.1　实验方案

本实验在 QEMU 模拟器[8]上运行 NOVA 系统及其运行时环境 NRE, 加载 I/O 访问控制配置文件, 并新建指定了安全等级标签的虚拟机. 我们通过在虚拟机上执行不同类型的任务, 记录并比较虚拟机在系统加载 I/O 访问控制模块前后、不同负载情况下的性能损耗情况, 对 I/O 访问控制模块对系统整体性能造成的影响进行分析.

4.2 I/O 密集型任务

I/O 密集型(I/O–bound)任务大部分时间用于执行 I/O(如硬盘)的读写操作, CPU 负载较低[9]. 本实验在系统加载访问控制模块前后, 分别在虚拟机上执行 linux

dd 命令进行磁盘块读写操作, 并通过 Linux time 命令测量执行时间. 通过 10 次实验取平均值, 得到加载访问控制模块前后相同 I/O 密集型任务执行时间的对比, 如表 2 所示.

表 2 I/O 密集型任务执行时间(单位: 秒)

	加载访问控制模块前	加载访问控制模块后
Real	27.13	27.39
User	1.13	1.19
Sys	15.47	15.65

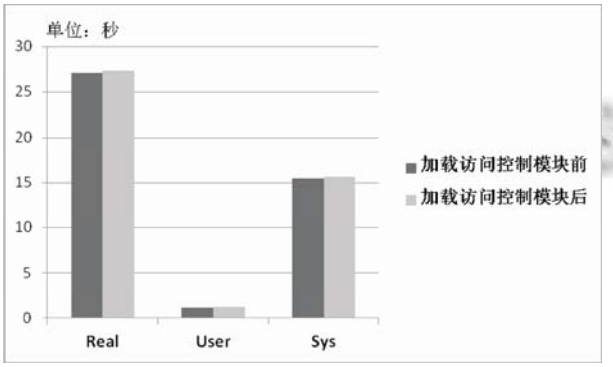

图 6 I/O 密集型任务执行时间对比

4.3 其他类型任务

为了验证 I/O 安全机制是否会对系统中其他类型的任务产生影响, 我们在虚拟机中运行计算密集型任务并记录执行时间. 计算密集型(CPU–bound)任务在执行过程中, 大部分时间用做计算与逻辑判断等 CPU 操作, 一般 CPU 占用率较高, 甚至接近 100%[10]. 本实验在系统加载访问控制模块前后, 分别在虚拟机上运行循环计算脚本, 使得 CPU 空闲率接近 0%, 并利用 Linux time 命令测量程序执行时间. 通过 10 次实验取平均值, 得到加载访问控制模块前后相同计算密集型程序执行时间的对比, 如表 3 所示.

4.4 实验结果分析

根据图 6、图 7 所示实验结果, 可以看出添加访问规则与模块前后, 虚拟机 I/O 密集型任务与其他类型任务, 如计算密集型任务的时间几乎不受影响. 原因在于: Vancouver 中的访问控制模块在拦截到虚拟机发来的 I/O 资源访问控制请求后, 只是根据当前虚拟机监控器可访问的权限表进行比对判断, 禁止违反访问权限表配置的请求继续被处理, 其占用的时间相较于任务实际执行时间可忽略不计. 因此添加虚拟机对 I/O 资源的访问控制并不会对系统整体性能造成影响.

表 3 计算密集型任务执行时间(单位: 秒)

	加载访问控制模块前	加载访问控制模块后
Real	45.23	45.33
User	36.43	36.73
Sys	8.19	8.83

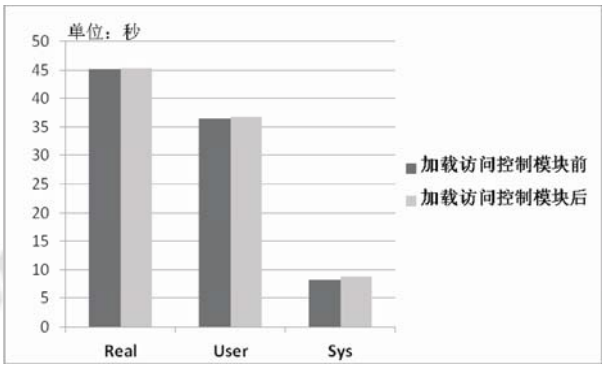

图 7 计算密集型任务执行时间对比

5 结论

本文首先简要介绍了基于微内核的虚拟化架构 NOVA, 并针对其目前在 I/O 方面存在的安全问题进行了分析, 提出了一种新的安全机制. 该安全机制包括: 为虚拟机添加安全等级标签、进行安全域的划分; 以访问控制矩阵的方式描述每个安全域可访问的 I/O 资源; 添加 I/O 访问控制模块, 通过跟踪虚拟机通过虚拟机监控器进一步和 NRE 中抽象 I/O 设备模块进行交互的代码路径, 根据访问权限表的设置判断是否需要切断数据的交换, 进而实现各安全域内虚拟机对 I/O 资源可定制的访问控制. 该机制实现了 I/O 资源在不同虚拟机间共享权限的控制, 限制了不同安全域的虚拟机可访问的 I/O 资源, 降低了因访问不当造成的安全隐患, 提高了系统的安全性. 最后, 实验证明了添加该 I/O 资源访问控制安全机制并未对整体系统产生明显性能损耗.

参考文献

1 Wang G, Ng TS. The impact of virtualization on network performance of amazon EC2 data center. Proc. of the 29th Conference on Information Communications. Piscataway. IEEE Press. 2010. 1163–1171.

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. ACM SIGOPS Operating Systems Review,

2003, 37(5): 164–177.

3 Steinberg U, Kauer B. NOVA: a microhypervisor–based secure virtualization architecture. Proc. of the 5th European conference on Computer systems. New York. ACM. 2010. 209–222.

4 Wright C, Cowan C, Smalley S, Morris J, Kroah-Hartman G. Linux security modules: general security support for the Linux kernel. Proc. of the 11th USENIX Security Symposium. Berkeley. USENIX Association. 2002. 17–31.

5 Tolone W, Ahn GJ, Pai T, Hong SP. Access control in collaborative systems. ACM Computing Surveys (CSUR), 2005, 37(1): 29–41.

6 Sailer R, Jaeger T, Valdez E, Caceres R, Perez R, Berger S, Griffin JL, van Doorn L. Building a MAC-based security architecture for the Xen open-source hypervisor. Proc. of the 21st Annual Computer Security Applications Conference. Washington: IEEE Computer Society. 2005. 276–285.

7 Liedtke J. Toward real microkernels. Communications of the ACM, 1996, 39(9): 70–77.

8 Bellard F. QEMU, a fast and portable dynamic translator. Proc. of the Annual Conference on USENIX Annual Technical Conference. Berkeley. USENIX Association. 2005. 41–41.

9 Tian C, Zhou H, He Y, Zha L. A dynamic MapReduce scheduler for heterogeneous workloads. Proc. of the 2009 Eighth International Conference on Grid and Cooperative Computing. Washington. IEEE Computer Society. 2009. 218–224.

10 Silberschatz A, Galvin PB, Gagne G. Operating system concepts. 9th ed. New York: Wiley, 2012: 112–114.