通过虚拟机间互相看护减少重复计算的方法

吴佐平[1],王海龙[1],王宏岩[1],肖 敏[1],王 玮[2],夏心锋[2]
(1. 北京中电普华信息技术有限公司,北京 100107;
2. 江苏瑞中数据股份有限公司,天津 300300)

摘 要:人们的办公环境越来越多地依赖虚拟机、虚拟桌面基础架构(virtual desktop infrastructure,VDI)等技术,如果将传统的个人计算机(personal computer,PC)安全软件直接应用于此环境,将造成极大的浪费。比如,在每台虚拟机上都安装防病毒、防火墙、防泄漏、加密等软件,造成在一个宿主机上,有多个相互隔离的虚拟机,跑着相同的软件,工作的内容也完全一样(分散方式)。文中讨论如何在一台宿主机上只跑一个同类安全软件,就对所有在此设备上创建的虚拟机,提供同样的安全服务(共享方式)。这种方法在不依赖于虚拟化产品供应商提供了虚拟化下层的接口(底层接口),而是靠虚拟机直接共享安全进程、并通过虚拟机之间互相看护的方式实现。性能接近接底层口方式、但远高于分散方式;投入却远低于底层接口。

关键词:虚拟机;数据安全;进程看护;进程冲突;云计算平台
中图分类号:TP31 **文献标识码**:A **文章编号**:1673-629X(2021)0062-05

A Method of Reducing Double Counting by Monitoring Each Other Between Virtual Machines

WU Zuo-ping[1],WANG Hai-long[1],WANG Hong-yan[1],XIAO Min[1],
WANG Wei[2],XIA Xin-feng[2]
(1. Beijing CLP Information Technology Co.,Ltd.,Beijing 100107,China;
2. Jiangsu Ruizhong Data Co.,Ltd.,Tianjin 300300,China)

Abstract:The office environment is increasingly dependent on virtual machine,VDI and other technologies. If the traditional PC security software is directly applied to this environment,it will cause a great waste. For example,installing antivirus,firewall,leak prevention,encryption and other software on each virtual machine will result in multiple isolated virtual machines running the same software and doing exactly the same work (distributed mode) on a single host. We discuss how to provide the same security service (sharing mode) for all virtual machines created on a single host with only one of the same security software. This approach does not depend on the virtualization product vendor to provide the interface of the underlying virtualization (hypersion API),but relies on the virtual machines directly share the security process,and through the virtual machine mutual care way to achieve. The performance is close to the API mode,but much higher than the distributed mode. The budget is much lower than the API mode.

Key words:virual machine;data security;process monitor;process conflict;openstack

0 引 言

随着虚拟化和云计算技术的快速发展,相应的计算环境产生的安全问题,越来越受到人们的重视[1]。由于虚拟机厂商、安全系统类型繁多,需求各异等特性,也造成相应的安全产品实现方法各异,难以统一管理、统一使用[2]。往往在每台虚拟机上都安装防病毒、防火墙、防泄漏、加密等(文中成分散模式),来自不同安全厂商的安全软件,实际提供给用户办公用的计算能力被大大压缩[3-4]。且各个安全软件之间本身就产生冲突,不能互相保护。用户的安全投入,不能逐步增加安全系数,反而被互相掣肘,重复投入、造成浪费。

人们需要一种方案,将使得各个安全产品不必重复安装、重复占用算力(文中成共享模式),且互相之间减少冲突,甚至可以相互看护、唤醒[5]。

文中尝试讨论如下内容:

(1)减少虚拟化系统中安全进程的冲突;

(2)在跨虚拟机间不同安全进程的互相看护。

文中模型表现出了良好的性能,并在实现的测试

收稿日期:2021-03-11

作者简介:吴佐平(1980-),男,电力工程:中级,研究方向为电力大数据应用研究;通信作者:王海龙(1984-),男,硕士,计算机工程:中级,研究方向为计算机发展研究。

中,有效地提高了经济效益。

1　相关原理概述

1.1　如何减少虚拟化系统中安全进程的冲突

进程的冲突多是由争抢资源造成的,主要包括争抢中央处理器(central processing unit,CPU)、随机存取存储器(random access memory,RAM)、硬盘输入输出(input/output,I/O)、网络带宽等,尤其是安全进程多数以系统权限运行,冲突更容易产生。同样的文件在不同的虚拟机中重复进行安全扫描、加解密、权限验证大量消耗算力的代码 8 等等。

没经过处理的虚拟机安全软件的资源占用情况,如图 1 所示。

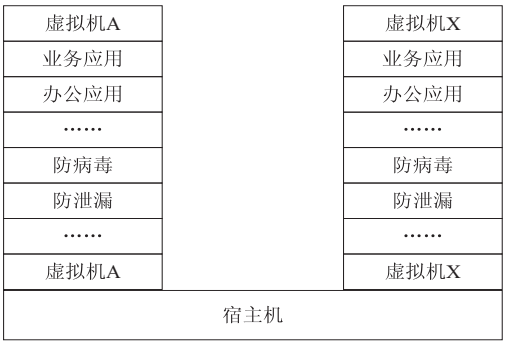

图 1　资源占用

如果将这些重复的安全进程移出去到同一个宿主中单独的虚拟机上,就能减少安全进程的资源占用。

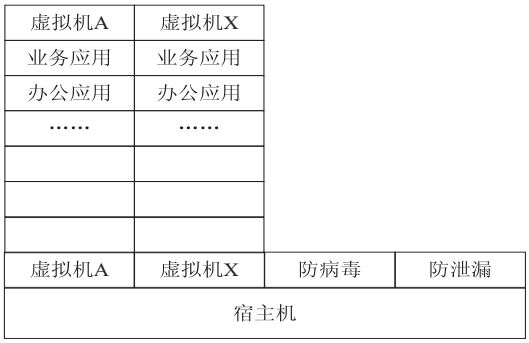

图 2　安全虚拟机

如图 2,即建立单独的安全虚拟机,将重复的安全进程从各个工作虚拟机上移到其上。这样在同一个宿主机上只保留一份安全进程拷贝。原来的进程直接调用,改成网络应用程序(Web Service)的方式进行。这样内存、硬盘的重复占用减少;由于各个虚拟机间共享扫描结果,不同虚拟机上相同文件的重复扫描也没有了,CPU 占用也减少。带宽占用虽然有所增加,但是,在同一个宿主机上,网络访问,可以通过虚拟化层,底层的接口通过内存拷贝实现。对外的网络带宽并不占用过多。

另外,不同安全进程不一定每个都占用单独的虚拟机,可根据需要共享,比如简单的入侵检测系统(intrusion detection system,IDS)或类似检测网际互联协议(internet protoco,IP)报文的安全产品。文中只讨论消耗扫描资源比较大的防病毒、内容识别的数据防泄露。

1.2　如何在跨虚拟机实现看护

一般考虑两种实现方法:

一种是通过调用虚拟化层,厂商提供的接口,将各个虚拟机中需要互相保护的进程,注册并通过宿主机看护,如图 3 所示。发现注册虚拟机上的进程死掉,将尝试重启该进程。虚拟机宕机,重启该虚拟机。

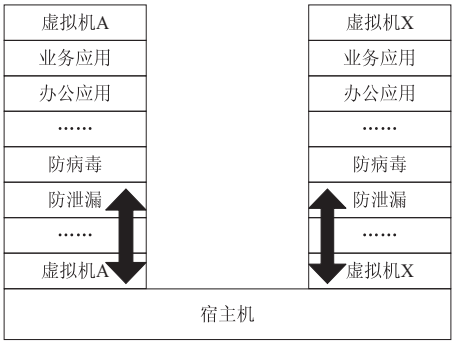

图 3　注册并通过宿主机看护表

另一种,创建一台虚拟机,把各个虚拟机上需要看护的进程注册到该虚拟机上。专门看护多个虚拟机上的进程,如图 4 所示。这个虚拟机可以和 1.1 中的某个负责安全的虚拟机共用,然后彼此注册,互相看护。

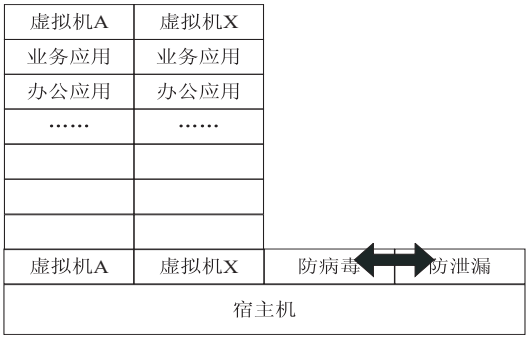

图 4　看护多个虚拟机上的进程

2　方案在 OpenStack 上的具体实现

调用云计算平台(OpenStack)监控接口开发,一般流程:

首先,完成身份认证[6];

其次,获取 OpenStack 计算节点列表即物理机(Hypervisor),并且获取虚拟处理器(VCPU)、内存、磁盘等信息[7];

最后,获取所有虚拟机实例(Instance)信息,或选择运行在某个计算节点物理机上的虚拟机实例信息[8]。

主要过程是,注册待看护进程名单,喂给进程数据,保证其启动并正常输出结果。

2.1 身份认证

所有的操作的前提都需要先完成身份认证。一般有两种方式：

```
配置文件认证
修改 auth. properties 修改验证信息：
authURL = http://controller:5000/v3/
domain = Default
username = admin
password = xxxxxxx
projectName = admin
```

使用认证工厂返回认证客户端实例,不为空则认证成功：

```
AuthService authService = new AuthFactory();
Assert. assertNotNull(authService. auth());
认证信息 Bean 认证
Authentication user = new Authentication();
user. setAuthURL("http://controller:5000/v3/");
user. setDomain("Default");
user. setProjectName("admin");
user. setUserName("admin");
user. setPassword("xxxxxxx");
AuthService authService = new AuthFactory();
Assert. assertNotNull(authService. auth(user));
```

2.2 管理物理机和虚拟实例

(1)获取所有计算节点物理机信息列表。

```
List<? extends Hypervisor> hypervisors = service. getHypervisors();
LOGGER. info("获取到如下物理计算节点:");
for (Hypervisor h : hypervisors) {
    LOGGER. info(h. getHypervisorHostname() + " " + h. toString());
}
```

(2)获取所有计算节点物理机资源使用情况的集合。

```
HypervisorStatistics statistics = service. getTotalStatistics();
LOGGER. info(statistics. toString());
```

2.3 开发看护进程并注册

(1)获取所有的虚拟机实例。

```
for (Server s : service. getAllInstance()) {
LOGGER. info(s. toString());
}
```

(2)通过计算节点 ID 获取运行在该计算节点上的实例。

```
boolean isHost = true;
for (Server s : service. getInstancesByHypervisorID("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")) {
    if (! s. getHostId(). equals("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")) {
        isHost = false;
    }
}
Assert. assertTrue(isHost);
```

(3)通过计算节点主机名获取运行在该计算节点上的实例。

```
boolean isHost = true;
for (Server s : service. getInstancesByHypervisorHostname("compute1")) {
    if (! s. getHypervisorHostname(). equals("compute1")) {
        isHost = false;
    }
}
Assert. assertTrue(isHost);
```

(4)获取所有的监控资源对象。

```
for (Resource r : service. getResourceList()) {
    LOGGER. info(r. toString());
}
```

(5)通过监控资源 ID 获取资源对象实例。

```
boolean result = false;
if (service. getResourceById("xxxxxxxxxxxxxxxxxxxxxxxx"). getId()
        . equals("xxxxxxxxxxxxxxxxxxxxxxxxxxxx")) {
    result = true;
}
Assert. assertTrue(result);
```

(6)获取所有的监控指标。

```
for (Meter m : service. getMeterList()) {
    LOGGER. info(m. toString());
}
```

(7)通过监控指标名称获取监控指标。

```
boolean result = true;
for (Meter m : service. getMeterListByName("memory")) {
    if (! m. getName(). equals("memory")) {
        result = false;
    }
}
Assert. assertTrue(result);
```

(8)资源 ID(实例 ID、镜像 ID 等)获取其所有监控指标。

```
boolean result = true;
for (Meter m : service. getMeterListByResourceId("xxxxxxxxxxxxxxxxxxxx")) {
    if (! m. getResourceId(). equals("xxxxxxxxxxxxxxxxxxxxxx")) {
        result = false;
    }
}
```

```
    Assert. assertTrue(result);
    (9)注册看护进程。
    boolean result = true;
    for (ProcessID m : service. setProcessprotect
("xxxxxxxxxxxxxxxxxxxx")) {
        if (! m. getProcessId(). equals
("xxxxxxxxxxxxxxxxxxxxx")) {
            result = false;
        }
    }
    Assert. assertTrue(result);
```

3　对比测试

3.1　实验设计

安装配置 OpenStack 环境,见表 1[9]。

表 1　OpenStack 环境

主机名	配置	IP 地址
Controller	CPU E5-2600 四核 RAM8G HD-76G	192. 168. 200. 101
compute	CPU E5-2680 12 核 RAM64G HD-2T	192. 168. 200. 102

设置第一组测试用虚拟机,用于测试各个虚拟机上安装重复软件的方案,总内存 40G,见表 2。

表 2　第一组测试

主机名	配置	IP 地址
办公 01	单核,4G,20G	192. 168. 100. 1
办公 02 ~ 09	单核,4G,20G	192. 168. 100. x
办公 10	单核,4G,20G	192. 168. 100. 10

设置第二组测试用虚拟机,用于测试各个单独配置安全虚拟机的方案,总内存 40G,见表 3。

表 3　第二组测试

主机名	配置	IP 地址
办公 01	单核 3G 20G	192. 168. 100. 1
办公 02 ~ 09	单核 3G 20G	192. 168. 100. x
办公 10	单核 3G 20G	192. 168. 100. 10
病毒扫描虚拟机	单核 3G 20G	192. 168. 100. 201
防泄露虚拟机	双核 7G 20G	192. 168. 100. 202

准备测试数据:病毒仿真码可执行文件、宏文件;内容识别待扫描的纯文本、富文本文件。按不同大小、类型分目录存放以便于性能测试数据分析,如图 5 所示。

准备测试工具:(1)办公仿真,脚本方式调用办公软件进行随机的增删改成操作后存盘[10];(2)批量内容识别脚本。

文中的实验结果均为独立重复 10 次实验之后的平均值。宿主机系统为 CentOS7 的 PC 上服务器,5 台虚拟机为 Windows7,5 台虚拟机为 Windows10,实验相关代码由 C/C++/Java 编写。

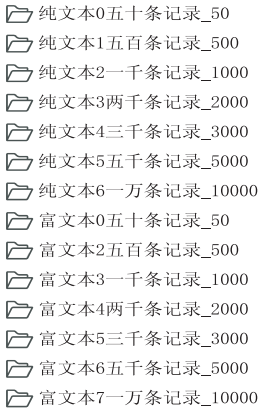

图 5　测试数据

3.2　性能分析

场景一:持续高频率扫描请求[11-13]。

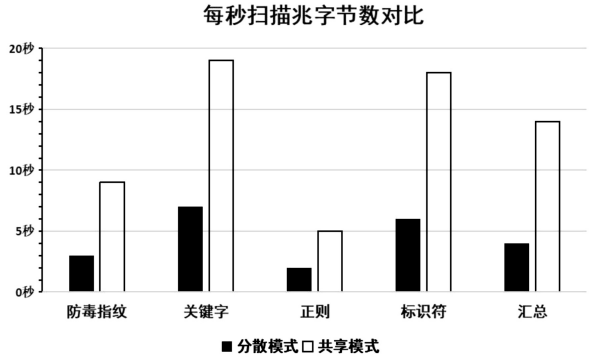

图 6　每秒扫描兆字节数对比

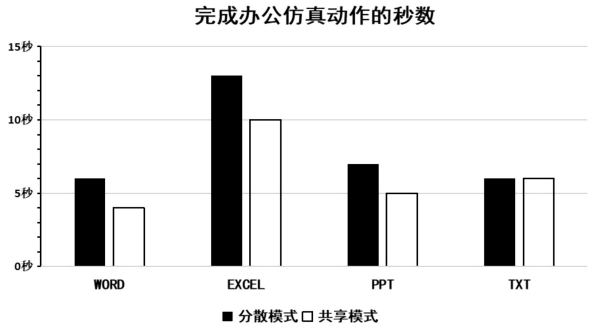

图 7　完成办公仿真动作的秒数

在此种场景中,共享模式的性能远高于分散模式。压力越大差距越明显。分析表明,虽然共享办公环境,单台机器内存小于共享模式,但由于将安全进程移除,实际运行时提供给办公用的资源反而更多。加大送扫压力,分散模式提前出现系统卡顿、数据崩溃等异常。共享模式的抗压能力,在此次测试中,平均比分散模式高出一两倍。尤其打开去重的选项时效果更佳明显,一些测试中共享模式达到分散模式的数十倍。

场景二:偶发日常扫描仿真[14-15]。

场景一采用的是固定送扫频率,持续送扫,多次测试逐渐增加,直到系统出现漏扫甚至崩溃。这样的场景在日常办公环境并不常发生。所以场景二,采用随机送扫,多数送扫频率低,偶然提高频率的方式。测试结果表明,两种模式的 CPU 使用性能类似。差别只是分散模式占用了较多的硬盘和内存。但是,它不用另外开发,原来单机上的程序拿来就能用。

4 结束语

共享模式在高并发扫描申请时有明显的性能、稳定性优势,但是需要一些开发成本。分散模式可以直接用单机版安装包,不增加开发成本,但是浪费了很多硬盘、内存自用。在高并发的环境下性能差、可靠性低。使用者应根据自己的运行环境和开发成本,选择适应自己的模式。

如果已经部署使用了 OpenStack 之类的云管理环境,采用共享模式应该是更合适的选择。

参考文献:

[1] 朱 民,涂碧波,孟 丹. 虚拟化软件栈安全研究[J]. 计算机学报,2017,40(2):481-504.

[2] 蔡建轩,李 梅. 基于 VMwarevSphere 的集群虚拟机安全问题研究[J]. 电脑知识与技术,2019,15(16):5-6.

[3] 唐建军,刘帅辰. IDC 虚拟化安全防护技术应用研究[J]. 中国新通信,2019(24):134-135.

[4] 吴兰华. 云计算虚拟化安全威胁及安全技术架构[J]. 电子技术与软件工程,2017(13):203.

[5] 王 勇. 基于虚拟隔离机制的云盘安全访问策略[J]. 信息与电脑,2016(5):201-202.

[6] 李海波,黄秋兰,石京燕,等. 基于 OpenStack 的虚拟资源调度技术研究[J]. 科研信息化技术与应用,2016,7(3):25-30.

[7] 殷童晶. OpenStack 虚拟云桌面系统的应用与实现[J]. 信息周刊,2018(30):214.

[8] 陈 凯. OpenStack 私有云的测试与评估[J]. 信息通信技术与政策,2016(12):49-52.

[9] 孔留彦. 分布式系统性能测试[J]. 信息与电脑,2017(11):68-71.

[10] 李茂斌,岳海燕. Java 调用 webService 应用[J]. 数字技术与应用,2017(3):203-204.

[11] 梁 毅,曾绍康,梁岩德,等. 一种基于周期性特征的数据中心在线负载资源预测方法[J]. 计算机工程与科学,2020(3):381-390.

[12] 陈继锋,刘 旭,张 丹,等. 基于迭代松弛的测试数据自动生成框架的分析与设计[J]. 湖南涉外经济学院学报,2010(4):16-17.

[13] 赵 晨,张淑萍. 透明进程间通信协议在集群系统中的应用[J]. 计算机工程与设计,2018(3):639-644.

[14] 罗恒洋,张 林. Java 中的正则表达式应用探讨[J]. 电脑知识与技术,2019(32):95-98.

[15] 骆惠清. 基于 Restful WebService 的家庭安全远程监控系统研究[J]. 延边大学学报:自然科学版,2017(3):252-254.

(上接第 46 页)

word problems on intelligent humanoid robot[J]. Procedia Computer Science,2018,135:719-726.

[4] MIYANI M,DOSHI S,JAIN J. Word problem solver system using artificial intelligence[J]. Procedia Computer Science,2015,45:800-807.

[5] 张 果. 面向初等数学应用题自动解答的核心技术研究与应用[D]. 成都:电子科技大学,2019.

[6] 潘 黎,冯 速. 基于概念层次网络的小学应用题句类分析和知识提取[J]. 计算机系统应用,2009,18(12):179-183.

[7] 吴林静,劳传媛,刘清堂,等. 基于依存句法的初等数学分层抽样应用题题意理解[J]. 计算机应用与软件,2019,36(5):132-138.

[8] 王 磊. 基于深度强化学习的数学应用题自动求解器[D]. 成都:电子科技大学,2019.

[9] 孙劲光,王 松. 数字相关特征的文字应用题深度学习自动求解[J]. 辽宁工程技术大学学报:自然科学版,2019,38(5):459-462.

[10] 邵 群. 基于知识库的应用题问题求解系统[D]. 北京:北京大学,1996.

[11] 王 军. 基于描述逻辑的语义 Web 知识表示和推理[D]. 桂林:广西师范大学,2008.

[12] JI Xiaonan,RITTER A,YEN P Y. Using ontology-based semantic similarity to facilitate the article screening process for systematic reviews[J]. Journal of Biomedical Informatics,2017,69:33-42.

[13] ABEND O,RAPPOPORT A. The state of the art in semantic representation[C]//Proceedings of the 55th annual meeting of the association for computational linguistics. Vancouver,Canada:ACM,2017:77-89.

[14] 张清华,幸禹可,周玉兰. 基于粒计算的增量式知识获取方法[J]. 电子与信息学报,2011(2):185-191.

[15] 王 利,张喜平,郭 林. 增量式知识获取算法综述[J]. 重庆邮电大学学报:自然科学版,2007,19(B06):99-102.

[16] 胡 晓,胡 洁,彭颖红等. 语义级知识融合中的冲突消解方法[J]. 上海交通大学学报,2009(11):55-58.