**海量存储技术课程报告**

|  |
| --- |
| **SGX性能优化新接口:HotCalls** |

|  |  |
| --- | --- |
| 院（系）名 称 ： | 计算机学院 |
| 专 业 名 称 ： | 信息安全 |
| 学 生 姓 名 ： | 王斐 |
| 学 号 ： | 2017202110086 |

二〇一七年十二月

# 摘 要

英特尔的SGX安全执行技术允许使用不可信服务器计算秘密数据，但是使用SGX所带来的性能影响仍然是个问题。文章通过综合定量地测试发现，直接使用SGX的库原语来调用函数会增加很大的开销，在系统调用频率高的应用中可能导致性能下降79%。因此利用特定于SGX操作的新微基准集来调查性能下降的原因，并由此设计出了一个新的SGX接口框架HotCalls。HotCalls利用的是同步自旋锁机制，容易集成。实验证明，它可以将吞吐量提高2.6-3.7倍，并将应用延迟减少62-74%。我将通过对HotCalls的深入理解分析，以及结合之前所参与过的相关开发经历来进行详细报告。

关键词：SGX；性能优化；存储硬件安全

# ABSTRACT

Intel’s SGX secure execution technology allows running computations on secret data using untrusted servers.While the performance implications of using the technology remains an open question. We show that straightforward use of SGX library primitives for calling functions add more overhead. In applications with high system calls frequency,the performance degradation may be as high as 79%. We investigate the sources of this performance degradation by leveraging a new set of microbenchmarks for SGX-specific operations. We leverage the insights we gain from these analyses to design a new SGX interface framework HotCalls. HotCalls are based on a synchronization spin-lock mechanism and can easily be integrated into existing code.Experiments prove that using the new interface boosts the throughput by 2.6-3.7x, and reduces application latency by 62-74%. I'll give the report through in-depth understanding of HotCalls and the related development experience I've been involved with before.

**Key Words**: SGX;Performance optimization;Storage hardware security

**目录**

[1 简介 1](#_Toc500613438)

[2 SGX 1](#_Toc500613439)

[3 基本操作的开销 2](#_Toc500613440)

[3.1 Ecalls的开销度量 3](#_Toc500613441)

[3.2 Ocall的开销度量 4](#_Toc500613442)

[3.3 内存访问的开销度量 5](#_Toc500613443)

[3.4 小结 6](#_Toc500613444)

[4 HotCalls:一个优化的SGX接口 6](#_Toc500613445)

[4.1 HotCalls架构 6](#_Toc500613446)

[4.2 实际考虑 7](#_Toc500613447)

[4.3 HotCalls的实证评估与安全评估 7](#_Toc500613448)

[5 在应用程序上评估HotCalls 8](#_Toc500613449)

[5.1 高效的应用程序移植 8](#_Toc500613450)

[5.2 具体评估 8](#_Toc500613451)

[6 相关工作 9](#_Toc500613452)

[7 总结分析与自我扩展 10](#_Toc500613453)

[参考文献 10](#_Toc500613454)

# 1 简介

云计算降低了计算和存储成本，但是这意味着你需要完全信任提供服务的第三方，同时还要信任操作系统、虚拟机管理器以及各种固件。

2015年英特尔发布了Skylake架构的处理器，该架构内置了一种新安全技术SGX(Intel Software Guard Extension),主要功能是在计算平台上提供一个可信空间，保障用户关键代码和数据的机密性和完整性。它将合法软件的安全操作和数据封装在一个enclave中，enclave是应用程序的地址空间中一个受保护的区域，安全边界只包含CPU和它自身，非enclave里的软件想访问其内存均会被阻止，即使是OS、BIOS和虚拟机监控器这样的特权软件。

近几年的工作定性讨论了在SGX enclave中运行应用程序的性能影响，但却没有说明是什么操作导致了性能下降以及下降了多少。该文首次对SGX进行了量化性能评估，对涉及到SGX框架的操作进行分类与开销测评。结果发现与常规的操作系统调用相比，涉及SGX的调用操作使性能下降了54-113倍。

另外，该文通过评估三种微基准来估计将缓冲区传出传入enclave的开销。Memory Encryption Engine（MEE）比基准操作增加了12%的开销；该文的微基准加解密使内存访问时间增加到了102%；在mcf和libquantum中则分别延迟了55%和420%。SGX相关调用的开销对于存在高频率系统调用的应用程序来说是个关键的瓶颈，它会浪费一部分内核时间。用于方便系统调用的环境切换也是SGX程序的主要瓶颈。基于这些分析，该文提出了一个用于enclave内与外部不可信区域的交互的高性能替代接口HotCalls，比SGX SDK快13-27倍。

最后在常用的应用程序上评估了HotCalls，包括：openVPN、memcached、lighttpd。证明了它可以改善吞吐量并减少延迟。

# 2 SGX

SGX指令集内置于英特尔第六代微处理器架构Skylake，它允许创建名为enclave的安全执行环境，类似于ARM Trust Zone，不同的是SGX只拥有用户级权限。enclave就像一个反向沙盒，开机时BIOS在内存中定义一个名为Enclave页缓存（EPC）的区域，它被MEE加密，是处理器保留内存（PRM）的一部分，不能被任何软件访问。

安全环境创建过程如下：用ECREATE指令初始化一个enclave；调用EADD和EEXTEND指令把内存页拷贝进enclave的加密内存，其中内存页将被哈希来生成一个enclave度量；最后调用EINIT指令完成度量。认证过程中，CPU使用相关主秘密信息对该度量进行签名并产生一个报告，将报告发送到远程客户端，客户端验证签名后，通过认证过程创建的安全通道提供需要enclave处理的数据。

**进入enclave：**图1描述了enclave的内外交互。enclave初始化后，外部不可信代码只能通过调用EENTER指令执行enclave内部的可信代码，该指令执行环境切换，保存不可信代码的状态，恢复可信代码的最后已知状态。英特尔提供了名为ecall的封装代码来执行环境准备与EENTER指令调用。

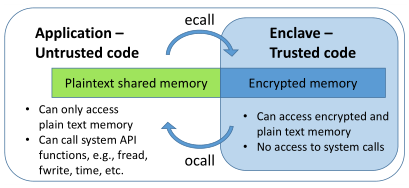


图1 enclave内外交互

**访问外部资源：**enclave只有退出到不可信区域才能访问硬件或操作系统资源，这可以通过EEXIT指令执行反向环境切换实现，封装后的代码名为ocall。

**声明边界调用：**英特尔提供了一个名为edger8r的边界函数构造工具，程序员需要在EDL扩展文件中利用特定于SGX的语法声明边界函数，包括函数接受的参数及其属性，然后edger8r解析EDL文件并生成ecalls和ocalls的封装代码。HotCalls也利用了这些代码来方便调用。

# 3 基本操作的开销

我在表1中总结了所评估的各种微基准，其中1,2,4,5是环境切换的开销；3,6是传递参数和数据的成本；7,8是对连续内存缓冲区的访问时间；9,10是非连续读写的访问时间。

该文使用读时间戳计数器指令RDTSCP来按时钟周期评估执行时间，该指令只能在enclave外执行。将每个微基准运行10组，每组20000次，共200000次测试。由于利用RDTSCP测量用户空间操作的周期数时，环境切换会导致异步退出从而使测量不准确，因此进行多次实验并舍弃被影响的部分。

表1 微基准评估

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Micro-benchmark** | **Description** | **Median Latency(cycles)** |
| 1 | Ecall (warm cache) | 调用一个没有参数的enclave并即刻返回，见图2a（实线） | 8,640 |
| 2 | Ecall (cold cache) | 同上，但测试前会刷新缓存，见图2a（虚线） | 14,170 |
| 3 | Ecall buffer transfer | 调用enclave函数，传递一个2KB缓冲区to / from / to&from enclave. 图4为其他大小缓冲区情况 | 9,861/11,172/10,827 |
| 4 | Ocall (warm cache) | 退出enclave执行不可信调用，见图2b（实线） | 8,314 |
| 5 | Ocall (cold cache) | 同上，但测试前会刷新缓存，见图2b（虚线） | 14,160 |
| 6 | Ocall buffer transfer | 调用不可信代码，传递一个2KB缓冲区to / from / to&from enclave. 图5为其他大小缓冲区情况 | 9,252 / 11,418 / 9,801 |
| 7 | Reading memory | 连续从加密/明文内存中的2KB大小的缓冲区中进行读取，图6为其他大小缓冲区情况 | 1,124 / 727 |
| 8 | Writing memory | 连续从加密/明文内存中的2KB大小的缓冲区中进行写入，图7为其他大小缓冲区情况 | 6,875 / 6,458 |
| 9 | Cache load miss | 从加密/明文内存中读取8字节 | 400 / 308 |
| 10 | Cache store miss | 往加密/明文内存中写入8字节 | 575 / 481 |

## 3.1 Ecalls的开销度量

进出enclave的转换一部分通过SDK软件实现，一部分通过EENTER和EEXI指令硬件实现。

ecall首先利用特定ID定位enclave，然后获取读写锁，找到一个可用的线程控制结构（TCS），保存高级矢量扩展（AVX），检查浮点异常，最后调用EENTER指令。EENTER指令执行安全环境转换。验证完SGX enclave控制结构（SECS）和TCS后，代表不可信环境的寄存器将被备份，enclave环境被载入。在enclave中执行完可信函数后，EEXIT指令执行反向环境转换。

这里用一个不接受和返回任何参数的空ecall来测其延迟。图2a显示，热缓存情况下99.9%的结果在8600-8700个周期之间。冷缓存情况下完整执行一次ecall的时间是12500-17000个周期，比OS系统调用慢了83-113倍。

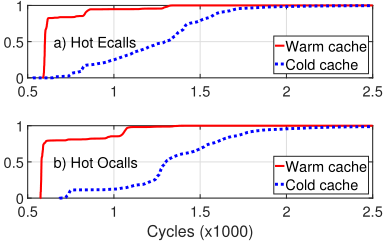
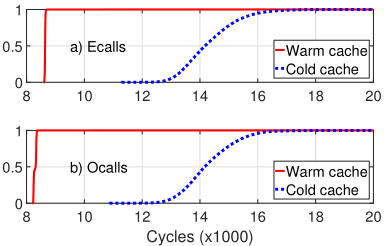


图2 ecalls和ocalls性能的CDF 图3 HotEcalls和HotOcalls的CDF

把参数传入安全区域时，SDK框架会在一个连续数据结构中序列化所有参数，该结构位于不可信内存中，然后把一个指向这个数据结构的指针传入可信区域。为防止数据从安全内存中泄露，需要确保整个被指向的数据结构都在enclave内存外。在EDL文件中声明ecall时，程序员处理缓冲区可以在以下四个选项中进行选择：user\_check,in,out,in&out。图4反映了ecall的全程时间。

Out选项中用到的memset函数按字节操作，在64位的平台上非常低效。in&out选项中用到的memcy函数比memset更高效，所以比out选项要快。

## 3.2 Ocall的开销度量

安全代码需要访问外部资源时就要调用ocalls。在EDL文件中声明ocalls，由edger8r生成封装代码，可信区域打包数据结构，对指针值执行安全检查，然后执行EEXIT指令。不可信区域整理输入参数，调用需要的OS API，打包需要返回给enclave的数据，然后执行ERESUME指令来恢复可信代码的执行。硬件接口执行EEXIT和ERESUME指令。

和ecalls一样，可以在EDL文件中标记指针参数来指示edger8r如何生成封装代码。图2b显示了执行一次ocall的全程时间。图5显示了传送缓冲区时使用四个选项的性能影响。

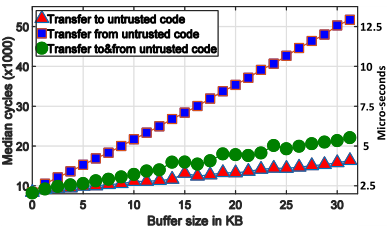
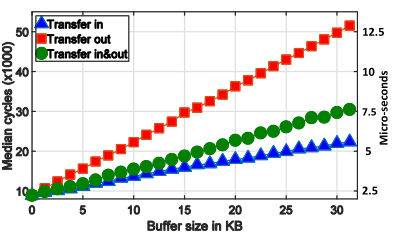


图4 ecall的全程延迟 图5 ocall的全程延迟

*user\_check*需要零拷贝且不需要安全检查，它在传递由OS提供的指针时有用。选择*in*时，封装代码验证指针指向enclave内，然后它根据enclave提供的大小参数在不安全堆栈上分配内存，最后把缓冲区复制到不安全内存。重新进入enclave时，被分配的内存通过展开不安全的栈被释放。选择*out*时，封装代码根据enclave提供的大小参数在不安全堆栈上分配内存，然后不安全缓冲区中新分配的内存被清零。ocalls返回后，缓冲区被拷贝回安全内存。该文认为清零不安全内存中的缓冲区没有安全意义且很低效，因此可以将此操作删除。

## 3.3 内存访问的开销度量

MEE为整个EPC提供机密性、完整性和抗回滚攻击保护，这是通过维护一个完整性树实现的。完整遍历这个树涉及了若干内存访问，因此“MEE cache”用于在访问附近内存地址时防止产生显著性能开销。图6,7描述了访问不同大小缓冲区加密和明文内存的内存读写时间。

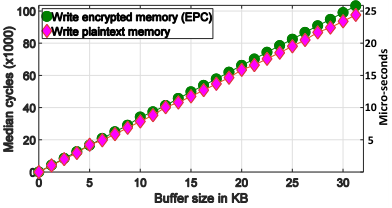
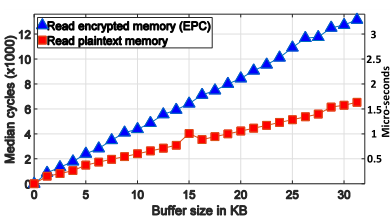


图6 读时间 图7 写时间

图8前四栏显示了该文微基准读写加密内存的结果。访问加密内存与明文内存相比，缓存加载缺失和缓存存储缺失分别慢了30%和19.5%。

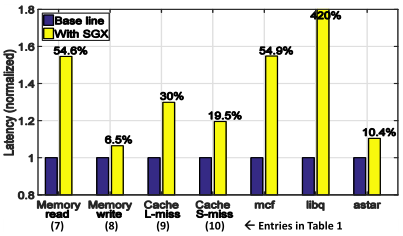


图8 内存加密开销

为了测试复杂内存访问模式，该文从SPEC 2006中选择了3个高内存密集型基准：mcf,libquantum,astar。图8对结果进行了比较，mcf在SGX enclave中运行减慢了55%，libquantum慢了5.2倍。其中libquantum变慢的原因可能是需要对加密内存页进行分页。

## 3.4 小结

**Ecalls和Ocalls的开销：**与常规OS系统调用相比，ecall周期数在热缓存情况下是其54倍，在冷缓存情况下是其83-113倍。HotCalls比默认的ecalls和ocalls快了13-27倍。另外，通过测试可以发现ocalls的执行比ecalls稍微快一些。

**内存访问的开销：**对加密内存的写访问引发了6.5-19.5%的开销，读访问是30-102%，具体取决于缓冲区大小。

**选择正确的传送方法：**out选项效率低，相比之下in&out更高效。

**User\_check的选择：**如果enclave只需要把数据转存到输出缓冲区，可以选择user\_check/zero-copy选项，让enclave直接把数据写入未加密内存来节省开销。

**进一步优化：**使用优化版本的memset和memcpy会更高效。

# 4 HotCalls:一个优化的SGX接口

HotCalls是执行ecalls和ocalls的替代机制，能带来一个数量级的性能提升。SGX调用依赖于安全环境转换，HotCalls操作基于使用共享非加密内存。

## 4.1 HotCalls架构

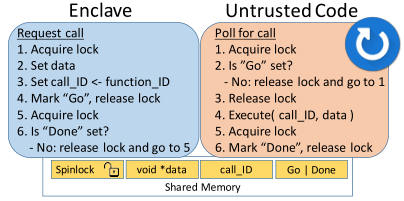


图9 HotCall架构

该架构由一个请求者和一个响应者组成，通过未加密的共享内存进行通信。图9展示了整个架构，其中enclave内代码是请求者，不可信区域代码是响应者。当请求者请求调用时会获取一个自旋锁，并检查共享布尔变量来确认响应者不处于繁忙状态。若响应者可用，请求者会拷贝相关代码到不可信共享内存，并将\*data指针指向它。在该数据结构中封装参数的代码与SDK ecalls/ocalls机制使用的代码相同。为了支持多个指定调用，请求者会指定所请求调用的ID，这是一个为响应者所知道的函数调用表的入口ID。一旦数据指针和请求调用ID到位，请求者会通过标记响应者为繁忙并释放锁来向响应者发出”go”指令。响应者会持续监控相同的共享内存并在被请求时执行相关调用。

## 4.2 实际考虑

**自旋锁:**SGX SDK提供了sgx\_spin\_lock，这是一个简单的繁忙-等待实现，因为与SGX无关所以enclave内外都能用。

**最小化自我争用:**为保证请求者和响应者均有机会获取到自旋锁，在释放锁之后会添加PAUSE指令。它能提高繁忙等待循环的性能，也能减少功耗。

**最大化利用率:**通过与若干请求者共享应答者线程来提高。

**防止饥饿:**请求者可以设置超时-检查响应者是否可用的最大次数，若发生超时则请求者可以回退去使用常规SDK调用这里设置为10。

**在空闲时节省资源:**设置一个超时计数器，在没有请求时进行递减，请求到达时则重置，当计数器到期时，响应者会设置sleep标志。

**打包传送给函数的参数:**SDK会生成打包ecall/ocall参数的代码，HotCalls会自动使用该代码来打包参数或拷贝缓冲区。这在保证安全的前提下避免了冗余的缓冲区清零。

## 4.3 HotCalls的实证评估与安全评估

图3显示了HotEcalls和HotOcalls延迟的CDF，在热缓存的情况下78%的调用少于620个周期。总的来讲，超过99.97%的调用都只需要少于1400个周期，而SGX SDK所提供的ecalls/ocalls机制需要8200-17000个周期。

该文通过检查enclave中可能影响可信代码运行的修改来评估使用HotCalls带来的安全影响。HotCalls用于打包数据结构的源码和SGX SDK一样，均由edger8r工具生成。任何操纵HotCalls使用的data指针的安全漏洞，对于被SDK的ecalls/ocalls实现所传送的指针来说也存在。HotCalls中的call\_ID相当于SDK中的ocall\_index变量，没有引入任何新漏洞。篡改自旋锁提供的同步性将会导致DoS攻击或多个线程同时访问明文内存中的相同数据，这与SGX不可信内存中指针带来的问题类似。

# 5 在应用程序上评估HotCalls

为了测试SGX在复杂软件上的性能影响选择了三个应用：memcached,openVPN和lighttpd。图10和11显示了这些应用在正常执行和在SGX enclave中执行的吞吐量和延迟。

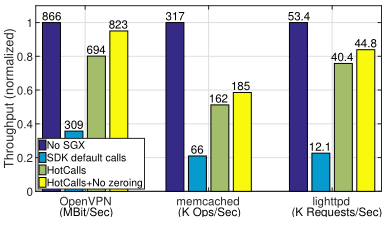
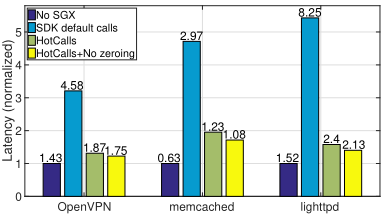
 

图10 吞吐量比较 图11 延迟比较

## 5.1 高效的应用程序移植

每个被调用的外部函数需要生成一个封装函数、一个ocall的EDL声明和一个返回函数。该文开发了一个框架来识别未定义的引用，它允许手动添加异常，表2中总结了最常出现的调用。pthread\_creat创建新线程时需要使用指向驻留在enclave内代码的指针，这样会导致调用失败，所以该文构造了一个新的ecall边界函数RunEnclaveFunction来完成pthread\_creat的功能。当使用到同步机制时都用SGX SDK相关架构替代。

表2 API调用数

|  |  |  |  |
| --- | --- | --- | --- |
| **Application** | **Frequent Calls(Calls x1000 / second)** | **Total Calls** | **Core Time** |
| Memcached | read(66.5), sendmsg(66.5),  RunEnclaveFucntion(66.5) | 200K | 42% |
| OpenVPN | poll(87), time(87), getpid(13.6),write(30), recvfrom(30),read(13.6) sendto(13.6) | 275K | 57% |
| Lighttpd | read(49),fcntl(25),epoll\_ctl(25), close(25),setsockopt(25), \_fxstat64(25),inet\_ntop(12),accept(12),  inet\_addr(12),ioctl(12),\_open64\_2(12), sendfile64(12),shutdown(12),writev(12) | 270K | 56% |

## 5.2 具体评估

通过每秒处理请求的数量和响应延迟来衡量memcached的性能。原始的memcached每秒处理316500个请求，平均延迟为0.63ms；把它放入enclave中每秒处理66500个请求，平均延迟增加到2.97ms；使用HotCalls的结果为每秒处理162000个请求，平均延迟为1.23ms；使用HotCalls并删除冗余清零的结果为每秒处理185000个请求，延迟为1.08ms，这比SGX SDK调用增加了2.8倍的请求处理并减少了64%的延迟。图10和11展示了这些影响。

把openVPN移植到enclave中可以保护加密和身份认证密钥，这里使用iperf3评估吞吐量。在openVPN信道中运行iperf3测得866Mbit/sec的TCP带宽。把openVPN放进enclave中后带宽减少到309Mbit/sec，往返全程的平均延迟为1.427ms，其中SGX环境切换浪费了57%的内核时间，对此可以在enclave内实现效用函数来提高性能。使用HotCalls后带宽增加到了694Mbit/sec，往返全程的平均延迟为1.873ms，这比之前的带宽增加了2.25倍延迟减少了60%；使用HotCalls并删除冗余清零后带宽增加到了823Mbit/sec，往返全程的平均延迟为1.747ms，这比之前的带宽增加了2.66倍延迟减少了62%。

最后使用http\_load评估lighttpd 1.4.41的性能。未经修改的lighttpd每秒能处理平均53400个页，平均响应延迟为1.52ms；放进enclave中后减少到了12100个，延迟变成了8.25ms；使用HotCalls的结果为每秒吞吐量增加到40400，平均延迟减少到2.40ms，吞吐量增加了3.3倍，延迟减少了70%；使用HotCalls并删除冗余清零的结果是吞吐量增加到44800即3.7倍，延迟减少到2.13ms即减少了74%。

# 6 相关工作

相似的技术总结见表3。前人关于SGX的工作总结见表4。尽管这些工作都提到了使用SGX enclave的性能影响，但都没有提供操作的具体度量。

表3 相似技术

|  |  |
| --- | --- |
| 相似技术 | 具体描述 |
| Sanctum | 开源处理器RISC-V的替代安全执行技术 |
| ARM Trustzone | ARM中允许安全区域OS和普通区域OS平行共存的安全执行方案 |
| SecureME | 保护用户空间的应用程序内存 |
| Overshadow | 提供相似保护的VMM强制机制，但只针对OS和应用程序 |

表4 相关工作

|  |  |
| --- | --- |
| 相关工作 | 具体描述 |
| Drawbridge | 把大部分OS操作转移到用户空间来改进安全性的方案 |
| Haven | 把整个应用及其OS-support需求移植到enclave中 |
| VC3 | 通过多服务器实现的多核处理，运行在SGX enclave上 |
| SCONE | 在enclave中运行docker容器，并提出改进容器性能的技术 |
| Ryoan | 提出一个架构允许一个enclave信任来自另一个提供者的enclave，并保证不会出现数据泄露 |

当然，也有一些相关的优化方案被提出。Linux NAPI用对事件进行轮询来代替调用，从而最小化对环境转换的需要。在虚拟环境下，这个方法被研究用来加速访问硬件或虚拟硬件，HotCalls就是受到这种方法的启发。

# 7 总结分析与自我扩展

这篇论文提出了一个SGX enclave的替代调用接口，将SGX的性能进行了提升。它可以优化调用延迟达13-27倍，提高吞吐量到3.7倍，并减少响应延迟到74%，它使得SGX更加安全实用。

在我看来，从TPM到TEE以及SGX，舍弃了TPM时代从BIOS到OS到APP整体可信，将保护的对象缩小到SGX时代的CPU本身以及某些APP的一部分内容。在提高性能的基础上，为存储安全提供了更多的可能。

我曾经参与开发过一个基于SGX的代理登录插件，其中SGX enclave最关键的功能便是用于存储秘密数据以及相关口令生成操作。然而当时我们发现它虽然提供了很强的存储与计算安全，但是性能确实有待提高。

通过对这篇论文进行详尽地阅读和理解，我对于SGX中的相关性能有了更加深入地了解，并知道了该从哪些方面入手去解决这些问题，对以后的学习有很大的帮助。

# 参考文献

1. Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, et al. Workload analysis of a large-scale key-value store[J]. ACM SIGMETRICS Performance Evaluation Review, 2012, 40(1):53-64.
2. Lehman T S, Hilton A D, Lee B C. PoisonIvy: Safe speculation for secure memory[C]// Ieee/acm International Symposium on Microarchitecture. IEEE, 2016:1-13.
3. Popa R A, Redfield C M S, Zeldovich N, et al. CryptDB: protecting confidentiality with encrypted query processing[J]. 2011:85-100.
4. Hunt T, Zhu Z, Xu Y, et al. Ryoan: a distributed sandbox for untrusted computation on secret data[C]// Usenix Conference on Operating Systems Design and Implementation. USENIX Association, 2016:533-549.
5. Sanjay Kumar, Himanshu Raj, Karsten Schwan,等. Re-architecting VMMs for Multicore Systems: The Sidecore Approach[C]// The Workshop on Interaction Between Opearting Systems & Computer Architecture. 2008.
6. Salim J H, Olsson R, Kuznetsov A. Beyond softnet[C]// Anual Linux Showcase & Conference. 2001.
7. Schuster F, Costa M, Fournet C, et al. VC3: Trustworthy Data Analytics in the Cloud Using SGX[C]// IEEE Symposium on Security and Privacy. IEEE Computer Society, 2015:38-54.
8. Dall C, Li S W, Jin T L, et al. ARM virtualization: performance and architectural implications[C]// International Symposium on Computer Architecture. IEEE Press, 2016:304-316.