Presented at the 4th ACM CCS Cloud Computing Security Workshop (CCSW'12), October 2012.

Towards a Richer Model of Cloud App Markets

Abhinav Srivastava
AT&T Labs-Research

Vinod Ganapathy
Rutgers University

ABSTRACT

Major cloud providers have recently been building cloud markets, which serve as a hosting platform for VMs pre-installed with a variety of software stacks. Clients of cloud computing leverage such markets by downloading and instantiating the VMs that best suit their computing needs, thereby saving the effort needed to configure and build VMs from scratch.

This vision paper argues for a richer model of cloud markets. We envision a market of VM apps that can interact with client VMs in a rich set of ways to provide a number of services that are currently supported only by cloud providers. For example, clients can use VM apps to deploy virtual machine introspection-based security tools and various network middleboxes on their work VMs without requiring the cloud provider to deploy these services on their behalf. This paper presents a taxonomy of VM apps, analyzes the key requirements needed to realize such VM apps, and explores the design and trade-offs of various options to implement VM apps.

Categories and Subject Descriptors. D.4.6 [**Operating Systems**]: Security and Protection
General Terms. Design, Experimentation, Management, Security
Keywords. cloud computing, security, app markets

1. INTRODUCTION

Infrastructure-as-a-Service (IaaS) cloud platforms such as Amazon EC2 and Windows Azure offer customers full access to virtual machines (VMs) whose software stacks they can customize and configure according to their needs. Enterprise-level clients with complex computing needs can clearly benefit from such flexibility. However, clients often have standard computing needs (e.g., they may simply want to host a Web server on the cloud), and may lack the resources and expertise to set up and configure VMs from scratch. This problem has motivated major cloud players to build *cloud markets* [4] to distribute VMs pre-installed with software stacks that address the needs of such clients.

In a cloud market, cloud providers or third-party developers build VMs customized for a variety of standard workflows, and publish images of these VMs in the market. Clients simply choose these VMs to get started with their computing needs, thereby providing a more agile and hassle-free cloud computing experience. For example, Amazon allows publishers to create and publicly offer VM images, known as *Amazon Machine Images* (AMIs) [1], that execute on EC2. AMIs that offer a variety of standard software stacks (e.g. LAMP, or SQL database software) are now available, which customers can directly instantiate to their specific domains. Publishers who create AMIs can also decide whether the AMIs must be paid or free.

In this paper, we argue that the above notion of cloud markets is nascent, and that cloud computing platforms can benefit from a richer model of "app stores" that are reminiscent of mobile app markets. Specifically, on current cloud markets, the notion of an app is restricted to VMs with different operating system versions, distributions, and other system or application software. These VMs are stand-alone, and do not necessarily cooperate or interact with each other to support complex operations. This is in contrast to the current model of mobile app markets, where apps can interact with each other, and even reuse code modules of other apps.

To illustrate the problem, consider that we want to offer security tools as a cloud-based service. That is, we would like to set up a VM equipped with standard security tools such as network intrusion detection systems (NIDS), firewalls, and sophisticated malware detectors, such as those based on virtual machine introspection (VMI) [10, 13, 14]. To benefit from the tools offered by this VM on current cloud markets, clients are required to install their software stacks within this VM. Instead, we aim for a model where this *security VM app* directly interacts with the client's work VMs (perhaps themselves downloaded from the cloud app market) to provide its services. For example, the security app VM must be able to monitor all incoming and outgoing network traffic from the client VMs, filtering this traffic using its NIDS and firewall.

We envision a *cloud app market*, where *VM apps* implement standard utilities such as firewalls, NIDS, storage encryption, and VMI-based security tools. VM apps can also implement a host of other non-security-related utilities, such as packet shapers, memory and disk deduplication, and QoS tools. Clients can leverage these utilities by simply downloading the appropriate VM apps, and *linking them suitably with their work VMs*. The key challenge in realizing this vision on current cloud computing environments is that such interaction between VMs is disallowed. Each virtualized platform has one privileged VM (also called the *management VM*), controlled by the cloud provider, that supervises the execution of client VMs. The management VM oversees all I/O from client VMs, and completely isolates VMs from each other. While such isolation is desirable across VMs of *different clients*, it also prevents VMs belonging to the same client from interacting in useful ways.

The goal of this paper is to expand on this vision of cloud app markets that support rich VM apps, and explore techniques for the implementation of VM apps. The contributions and outline of this paper are as follows:

- in Section 2, we present a taxonomy of VM apps, ranging from simple standalone VM apps, to ones that involve complex interactions with other VMs.
- in Section 3, we discuss the key requirements to enable the kinds of VM apps discussed in our taxonomy and motivating examples.
- in Section 4, we outline the design space of available options to implement these requirements.

CCSW'12, October 19, 2012, Raleigh, North Carolina, USA.

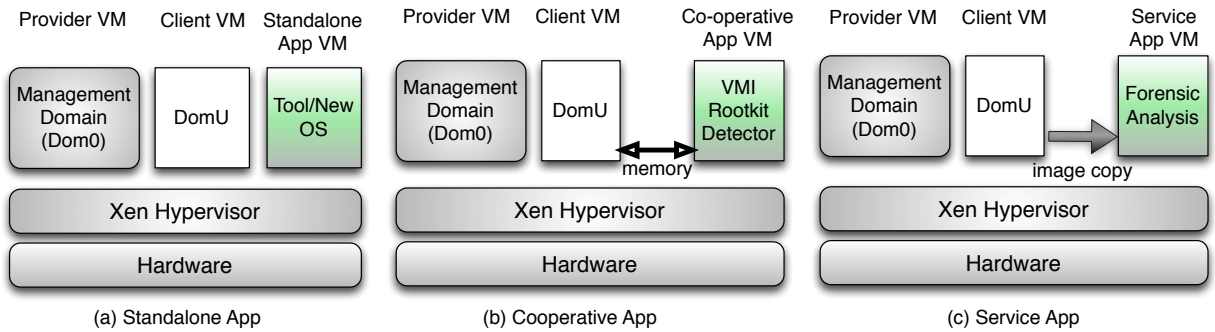


Figure 1. A taxonomy of VM apps

1.1 Example VM Apps

We now present two examples to demonstrate how security services can be implemented as VM apps.

VMI-based Intrusion Detection. To protect their work VMs from malicious software, clients of cloud computing can install anti-virus software within their VMs. Such an approach can detect malicious software that operates at the user level. However, attackers are increasingly exploiting kernel-level vulnerabilities to install rootkits, which in turn allow them to remain stealthy and retain long-term control over infected machines.

To counter kernel-level malware, researchers have proposed a number of solutions (e.g., [12, 10, 13, 14]) inspired by virtual machine introspection (VMI). In such solutions, the virtual machine monitor acts as the trusted computing base, and helps maintain the integrity of VMs that it executes. The security policy enforcer (e.g., a rootkit detector) itself operates within a privileged VM (e.g., the management VM), and checks the state of a target VM. To enable security enforcement, the virtual machine monitor fetches entities from the target VM (e.g., memory pages containing code or data) for inspection by the security policy enforcer. This architecture protects the enforcement engine from being tampered by malicious target VMs.

However, VMI-based solutions proposed to date are ideally suited for desktop virtualization because they require the security monitor to be installed within a privileged VM. While this is trivially possible in desktop virtualization environments, it cannot be implemented in the cloud because it requires cooperation from cloud providers.

In our approach, VMI-based intrusion detection tools such as rootkit detectors can be implemented within a VM app. To use the app, a client simply downloads the app, instantiates it, and permits the app to inspect the memory of its work VMs to detect attacks. Our VM app model therefore permits clients to assign specific privileges to its VM apps, such as mapping memory pages of its work VMs into the VM app's address space.

Network Security. Enterprises typically deploy a number of network security tools, such as firewalls, intrusion detection systems, and intrusion prevention systems, at the perimeter of networks to defend their in-house networks from attacks such as intrusions, denial-of-service (DoS), spam, and phishing, etc. Such deployment typically happens in the form of middleboxes. When an enterprise shifts its operations to public clouds, it must rely on the cloud provider to implement similar services on its work VMs. VM apps provide enterprises the freedom to implement these services without having to rely on cloud providers.

As in the previous case, the enterprise simply downloads the network security VM app, instantiates it, and configures it to receive both incoming and outgoing network flows to its work VMs. This would allow the enterprise to define and enforce arbitrary security policies flexibly, without rely on the cloud provider.

2. A TAXONOMY OF VM APPS

In this section, we present a taxonomy of VM apps based upon their functionality. We categorize VM apps into four groups: (1) standalone, (2) cooperative, (3) service, and (4) bundled.

2.1 Standalone VM Apps

Standalone VM apps (Figure 1(a)) represent the kind of apps that are currently available on cloud markets (e.g., AMI images available currently on Amazon's cloud market). These apps could include VMs with new operating system distributions, versions, or drivers supporting new hardware. Clients select VM apps, instantiate them, and build software and services atop the provided environment. These VM apps are self-contained and do not interact with the client's VM.

To create a standalone VM app, a publisher configures and installs an OS along with one or more user-level tools customized for a specific workflow. For example, a web-server VM app will likely contain the entire Apache toolchain. Likewise, it is also possible to imagine a security VM app that is pre-installed with user-space anti-virus tools. Clients that purchase such a VM app will benefit from the pre-installed anti-virus tools, thus saving them the effort of installing and configuring these tools themselves. They can simply install their enterprise software, which will automatically be protected by the anti-virus tools. Note that in the setting of standalone VM apps, a web-server VM app does not directly benefit from a security VM app because these VMs cannot interact with each other.

2.2 Cooperative VM Apps

As noted above, standalone VM apps do not interact with each other. Our vision of cloud app markets includes VM apps that can cooperate with each other. Such *cooperative VM apps* (Figure 1(b)) contain specialized software to be used on other client VMs (or VM apps), and actively communicate with the client VMs. For example, a checkpointing VM app will cooperate with a client VM to generate a snapshot of that VM. A cloud platform that supports cooperative VM apps will allow the creation of VMI-based intrusion detection apps, as outlined in Section 1.1. Since a cooperative VM app interacts with other client VMs, often to perform privileged operations (e.g., checkpointing its state or inspecting the contents of its memory pages) the client must assign specific privileges to it after instantiating it. The hallmark of cooperative apps is a two-way communication between the app and the client's work VM. For example, the VM app may send a request to the client VM to access

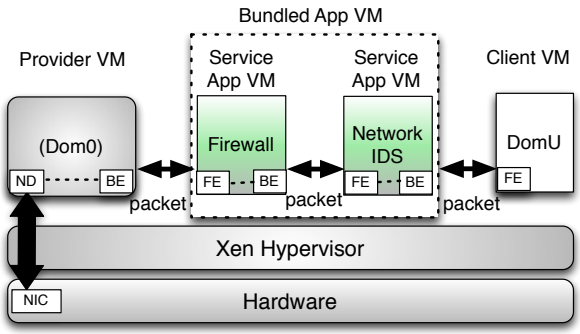


Figure 2. A bundled app VM along with a custom I/O channel connecting the client's work VM with the app VMs. FE stands for frontend, BE for backend, ND for network driver, and NIC for network interface card.

to a particular page, and the client VM would respond by giving it suitable access.

2.3 Service VM Apps

Service VM apps (Figure 1(c)) are those that are configured with specific tools and software that clients can use on their work VMs. Unlike cooperative apps, in which there is two-way communication between the VM app and the client's work VM, service VM apps operate on the output of the work VM. This category includes, for instance, forensic analysis VM apps that can operate on the memory snapshot of the client VM to conduct their analysis [15]. It also includes network security VM apps, where the network output of a VM may need to be examined by the VM app. To use service VM apps, clients set up the I/O path so as to direct the output of their work VM. Note that unlike standalone VM apps, which clients typically configure or extend, service VM apps are pre-installed with specific functionality (such as forensic analysis) that is executed on *other VMs*. Also, unlike cooperative VM apps, service VM apps do not require specific privileges to act upon other VMs because they only operate on the output of client VMs (e.g., VM images or network output). As will be explained in Section 3, to use service VM apps, clients require the flexibility to create custom I/O channels.

2.4 Bundled VM Apps

It is often beneficial to combine the functionality of multiple VM apps, and obtain a bundled VM app that composes the functionality of its constituent VM apps (Figure 2). Akin to the `pipe` primitive in operating systems, where the output of one command is input to another, in a bundled VM app, the output of one VM app is input to another. Individual VM apps inside a bundle could be standalone, cooperative, service VM apps, or themselves bundled apps.

Bundled VM apps are an ideal strategy for implementing and composing network services. For example, a client interested in network security apps, such as intrusion detection systems and firewalls, downloads a network security app bundle. Upon instantiation of the bundle, all the service VM apps inside it will create a chain of services. In this case, the client VM's packet will traverse the firewall, and then the network IDS, providing multiple layers of security. To realize the bundled VM app model, clients must fulfill the requirements of individual VMs in the bundle. For example: a bundle consisting of a cooperative and service VMs will require both necessary privileges and custom communication and I/O channels to carryout its task.

3. REQUIREMENTS

In this section, we describe key requirements that must be satisfied to realize our vision of cloud app markets, and the kinds of VM apps discussed in our taxonomy.

3.1 Trustworthy Launch of VM Apps

A common requirement, irrespective of the type of VM app used by clients, is the ability to perform trusted boot of the VM app. That is, the client must be able to verify that the VM app booted correctly, with the same software stack as promised on the cloud market. This requirement is particularly important because clients will typically download VM apps from the cloud market based upon the advertised functionality of the app. Because the VM app will likely interact with the client's work VMs, it is important for the client to validate that the operating system and applications running atop the VM app are as advertised by its creator. Such validation will establish trustworthy launch of VM apps.

We expect that validation of trustworthy VM app launch can be performed using trusted computing technology. We assume that a hardware root of trust, such as a *Trusted Platform Module* (TPM) chip is available on cloud hardware. Measurements are stored inside the TPM's Platform Configuration Registers (PCRs) and recursively obtained by *extending* the PCR's contents with hashes of the software stack. A client can verify trustworthy launch by verifying the measurement received from the platform using the expected hashes of the VM app.

3.2 Privilege Model

On today's cloud platforms, the provider maintains full control over the computing platform. The management VM of each virtualized hardware platform controls access to physical hardware resources. Clients can modify their own VMs at will, but these VMs are themselves not privileged.

Services such as checkpointing, VM creation, and VM intros pection-based security tools execute within the management VM. This domain is bestowed with privileges to inspect the memory contents and control I/O of client VMs. Since the management VM is under the control of providers, clients have to rely on cloud providers to adapt new tools and technologies and deploy them in the management VM.

Our proposed cloud app market model aims to remedy this situation by putting clients in charge of deploying services on their own VMs. To realize this vision, we must carefully consider the privileges that VM apps would need. Consider for instance a cooperative VM app, which interacts with other VMs belonging to the client. As an example, VMI-based introspection app would need privileges to map memory pages of the client VM that it monitors. Such privileges are normally available only to the management VM. Since VM apps execute as user VMs, the privilege model of the cloud must be suitably modified to allow such operations. Further, this privilege model must be fine-grained, i.e., clients must be able to specify that a VM app be given specific privileges (over its memory pages, or I/O channels) over client VMs.

As we will discuss in Section 4, it is possible to realize this privilege model in a number of ways, either by modifying the hypervisor, or using nested virtualization.

3.3 Preventing Information Leakage

VM apps are expected to interact with client VMs, which may store and manipulate sensitive data. It is therefore crucial to ensure that VM apps do not leak sensitive data to their publishers or other malicious third parties. Controlling information flow and preventing information leaks is a problem with a rich history, and a number of techniques have been developed in the research literature. We

examine the potential for information leakage in various classes of VM apps.

Standalone VM apps typically contain software stacks for specific workflows. Clients can possibly extend standalone VMs to build full-fledged work VMs to achieve their computing needs. It is difficult to reason about information flow within standalone VMs, because controlling the flow of information in such VMs will likely require mechanisms both within the VM (e.g., to track sources of information flow, which may be domain-specific), as well as within the virtual machine monitor.

Cooperative and service VM apps interact with client VMs, and can therefore access sensitive client information. However, since these VM apps are specialized (unlike standalone VM apps, which can double as work VMs), clients can regulate how these VM apps externalize the data that they read from their work VMs. For example, the client could disallow a VMI-based intrusion detection system from accessing the network or persistent storage, thereby preventing it from sending any sensitive data that it may read from the client's VMs. On Xen, for instance, such control can be implemented using Xen grant tables. I/O on Xen happens via page-sharing: a VM that wishes to communicate with the outside world shares pages with the management VM to pass data to hardware. Xen grant tables can be used to specify that a particular VM should never be able to establish shared pages with the management VM, hence restricting I/O.

We expect that such information flow controls would likely mitigate the risk of leakage of sensitive information. However, completely eliminating the risk of information leakage is still challenging, e.g., Bugiel et al. [7] show that some cloud VMs store sensitive client data and do not erase it even when clients release the VM. To eliminate such threats, cloud infrastructure providers must offer image-cleaning services and control mechanisms, as proposed by Mirage [16].

3.4 Featherweight VM Apps

Standalone VM apps may often come pre-installed with full-fledged operating systems and user-level tools. This is because of the expectation that clients will typically extend these apps to create work VMs.

However, cooperative and service VM apps are often developed for a specific purpose, and need not run full-fledged operating systems and associated user-level utilities. For example, a cooperative app VM to create checkpoints need not run a complex operating system such as Linux or Windows, and could instead utilize kernels such as MiniOS [3] and TinyLinux [5] along with the user-level software supporting the snapshotting service. We refer to such apps as *featherweight VM apps*.

Featherweight VM apps will likely consume fewer resources on the cloud (memory, CPU cycles, I/O), and will therefore be more cost-effective to clients. This is key because on cloud markets, clients must pay *both* during the purchase of the VM app, as well as to execute the app on the cloud.

3.5 Standardized API Interface

Cloud providers such as Amazon and Rackspace offer API SDKs to aid the development of smart phone apps that access cloud resources. There are currently no such APIs for cloud app developers. To design the apps described in Section 2, developers will need support from cloud providers. For example, a VM introspection cloud app needs to interact with the cloud hypervisor to map client VM's memory pages. Although APIs exist for this purpose (e.g., XenAccess [13], LibVMI [2]), access to them is restricted to the management VM. We need to provide similar interfaces to cloud app developers. Cloud providers must expose some basic primitives to allow VM app developers to build using those primitives. An API SDK would be the best way for providers to expose their infrastructure to app developers.

Several classes of VM apps may also require a query interface. For example, a forensic VM app should be able to alert clients about the results of a scan of a memory image. Similarly, a network IDS VM should be able to disclose its findings to the client. Given that each VM app implements different services, querying and obtaining results from each VM app will likely place a huge additional burden on clients.

We therefore posit that VM apps must also expose a standardized query API. Cloud providers can publish this API, thereby allowing clients using VM apps that conform to this API to develop standard tools to interface with these apps. The nature of the query results still depends on the service implemented by the VM app. For example, a checkpointing service will likely return a VM image, while an anti-virus will return a file with the results of a scan. Nevertheless, clients often have common requirements that can be standardized as part of this API. For example, clients may require a secure interface to retrieve query results; therefore, the API could support a cryptographic interface to communicate with VM apps.

3.6 Plumbing I/O

Virtualized architectures, such as Xen and KVM, offer both para-virtualized and emulated devices. On para-virtualized platforms, the split driver model runs a backend driver in the management VM and a frontend driver in each client VM. For unmodified guest VMs (i.e., no para-virtualization), the management VM runs code to emulate all guest I/O with devices. Device emulation induces significant overhead, thereby driving most major operating system vendors to create para-virtualized versions.

Virtualization-based public cloud environments direct all client VM I/O through the management VM. This architecture therefore does not allow clients to manipulate their I/O in arbitrary ways. For example, a client cannot direct its traffic to a firewall VM app before sending it to the management VM. Similarly, the I/O architecture does not allow disk reads/writes to pass through a storage IDS app before going to the disk.

To support a diverse set of VM apps, cloud providers must allow clients to customize the plumbing of the I/O paths to achieve desired goals. For example, clients should be able to advertise a network firewall VM app as the backend for their work VMs, so that all work VM packets traverse the firewall. However, packets must reach to the real network interface card, which is under the control of the management VM. Therefore, the backend driver executing within the management VM must serve I/O requests from the VM app. This setting would allow packets from the VM app to traverse via the management VM and then outside the machine (as shown in Figure 2).

3.7 Migration

Migration of a VM involves moving a running VM from a source physical hardware platform to a target platform. This operation can be performed via suspend and resume or on "live" VMs [9]. A standalone app VM is also a client's work VM, hence existing mechanisms [9] developed for migration also apply to such VM apps.

However, the other classes of VM apps require the development of novel migration techniques. Cooperative, service, and bundled VM apps operate upon other VMs belonging to the client. If the cloud provider decides to migrate a client VM that is being serviced by a VM app, both VMs would have to be migrated to maintain the service. For example, a network firewall VM app protects a client's work VM by inspecting incoming and outgoing flows. If the VM app is not migrated along with the work VM, the work VM will execute unprotected during that period. To be able to provide the

Design	Performance	Deployability	Capability
VMM changes	Best	Poor	Best
Nested virtualization	Good	Good	Good
Paravirt-based nesting	Good	Best	Good

Figure 3. Comparing the merits of the three design options.

complete security, both app and work VMs need to be migrated together. Note that a single VM app (such as a firewall) may service multiple client VMs, only some of which may be migrated; in such cases, the VM app may need to be replicated on both the source and target platforms.

4. DESIGN SPACE

In this section, we explore various design options to realize our vision of rich VM apps. As discussed in the requirements section, VM apps must have the ability to perform privileged system tasks on client VMs. Stock virtual machine monitors do not grant user domains such privileges. Our designs therefore either require modifications to virtual machine monitors, or support for nested virtualization. We evaluate our designs with respect to three metrics:

(**1**) *Performance.* What is the runtime performance overhead imposed by the design? Note that the use of VM apps may intrinsically reduce the performance of client VMs, e.g., the use of a NIDS or firewall VM app will necessarily reduce the throughput of a network-bound client VM because of the additional network element introduced in the I/O path. In evaluating performance, we ignore this intrinsic overhead. Rather, we focus on the overhead introduced by the implementation technique used to build VM apps.

(**2**) *Deployability.* Does the design require invasive changes to virtual machine monitors, as deployed by cloud providers today?

(**3**) *Capability.* How easily can the design realize the kinds of VM apps described in Section 2?

Figure 3 summarizes the merits our designs using these metrics.

4.1 Virtual Machine Monitor Modification

Perhaps the most straightforward approach to realizing rich VM apps is to modify the virtual machine monitor (VMM). These modifications will primarily address the way in which the VMM assigns privileges to client VMs. On commodity VMMs, the management VM runs at the highest privilege level, and orchestrates all communication between VMs running as user domains. In this model, communication between VMs belonging to a single client goes through the management VM.

Our proposed modifications to the VMM will change its privilege model, and allow VMs belonging to a single client to delegate specific privileges to VM apps, e.g., it will allow a cooperative VM app (executing with the client's credentials) to map memory pages belonging to the client's VMs. Further, cloud platforms do not permit clients to advertise a VM other than the management VM as an I/O backend of their work VMs. To allow this, we would need to implement changes in the VM instantiation protocol, to take input from clients on their preferred I/O backend (default would be the management VM). This design would permit clients to plumb I/O paths of their VM at will, thereby allowing the implementation of service and bundled VM apps.

We have realized these modifications in Xen in a recent work [8]. That work introduces a new *self-service cloud computing* model, in which each client's VMs are logically grouped together into a single *meta-domain*. Each meta-domain has its own administrative interface, which in turn can assign privileges to individual VMs to communicate with each other. We implemented these changes in Xen using the Xen Security Modules (XSM) framework. XSM is a generalized mandatory access control interface that allows the creation of custom security functionality within Xen. It does so by placing a set of hooks at specific locations within Xen, and offering the flexibility of customizing their implementation. Extensions to Xen's privilege model can simply be implemented as a code module containing callbacks to be invoked by specific hooks. See Figure 4(a).

We now evaluate this design using our three criteria:

(**1**) *Performance.* We expect the runtime performance impact of this approach to be minimal. The performance overhead will likely stem from the impact of executing additional checks, e.g., as implemented in the XSM module that implements the privilege model. VM apps and client VMs continue to execute directly atop the modified VMM, and their performance will likely be comparable to VMs executing atop an unmodified VMM.

(**2**) *Capability.* The VMM can easily be modified to admit the implementation of all four kinds of VM apps described in Section 2. In particular, it eases the creation of bundled VM apps, because clients can plumb I/O paths of their VMs at will, thereby allowing the output of one VM app to be fed as input to another. Therefore, this design option allows us to realize our envisioned model of rich cloud app markets.

(**3**) *Deployability.* Unfortunately, this design option is also the most difficult one to deploy. Because it involves VMM modifications, existing cloud providers must be willing to adopt the required changes to their platforms.

4.2 Nested Virtualization

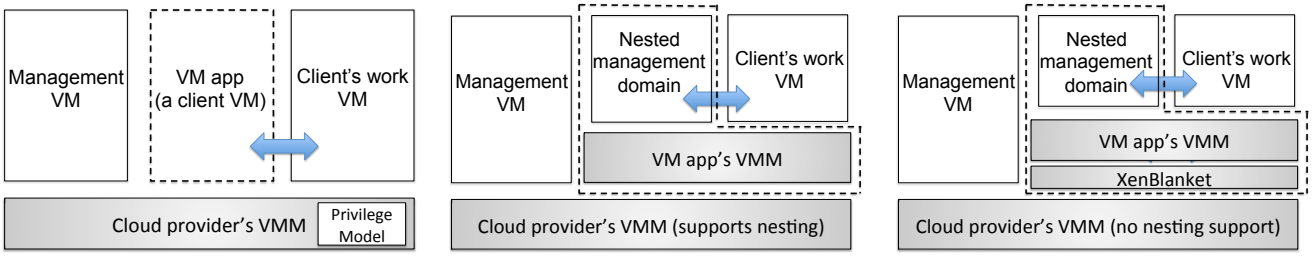
Hardware support for virtualization, coupled with software-level implementation tricks, have made low-overhead nested virtualization a reality [6]. Nested virtualization allows clients to execute a full-fledged VMM (a nested VMM) atop a base VMM that has been modified to support nesting. Clients can execute their work VMs atop the nested VMM, which in turn has its own management VM. Thus, clients can implement privileged services within the management VM of the nested VMM without relying on the cloud provider to do so. In this model, the VM app will be distributed as a VMM together with a management VM that implements the advertised functionality of the app. The client executes his work VM atop this VMM, which would itself execute on the cloud provider's VMM. See Figure 4(b).

We now evaluate this design using our three metrics.

(**1**) *Performance.* We expect the performance overhead of VM apps implemented using this technique to be acceptable, although the overhead will be higher than the previous approach involving VMM modifications. The primary source of overhead is that traps from the client VM must be handled both by the base VMM and the nested VMM. This overhead increases exponentially with the number of nesting levels (see [11, Section 3]).

(**2**) *Deployability.* This design is easily deployable over cloud platforms that support nested virtualization. For example, this design is directly applicable to OpenStack-based cloud platforms that are built atop KVM.

(**3**) *Capability.* This design easily accommodates the first three kinds of VM apps described in Section 2. However, it does not support easy creation of bundled VM apps. For example, suppose that we implement two VM apps, one to create VM checkpoints, and a second one to encrypt disk storage. In the nested VM app design, it is not as straightforward to compose these VM apps to build a bundled app that creates encrypted checkpoints. This is because because the services are implemented within the management VM of the nested VMM. Creating a bundled VM app would involve com-



(a) VMM modification **(b) Nested virtualization** **(c) Paravirtualization**

Figure 4. The design space of implementation options for VM apps. In each case, the components of the VM app are demarcated using dashed lines.

posing the functionality of the user-level utilities within the nested management VM.

4.3 Paravirtualization-based Nesting

Our third design option is a variant of the nested virtualized-based design described above. If the cloud provider's platform is based upon a VMM that does not have hardware support for efficient nested virtualization, it may still be possible to achieve many of the same benefits using paravirtualization. In particular, the XenBlanket project [17] demonstrates that it is possible to achieve nested virtualization atop a stock Xen VMM (e.g., as deployed on Amazon EC2) that does not support virtualization. XenBlanket achieves this by building a thin software layer (a "blanket") that emulates privileged operations for a nested VMM executing atop the base Xen VMM.

VM apps based upon this design option will resemble those developed for the previous design option. The principal difference is that the software stack of these VM apps and the client VMs must use a paravirtualized operating system. See Figure 4(c). Let us now evaluate this design using our metrics.

(**1**) *Performance.* XenBlanket was primarily developed to allow nested virtualization atop commodity cloud platforms. Although the reported overhead of the blanket layer is acceptable, the overheads of a VM app implemented using this approach will likely be higher than if support for nested virtualization was available. Among our design options, we therefore expect this approach to have the highest runtime overheads.

(**2**) *Deployability.* XenBlanket was developed with the primary aim of easy deployment over commodity clouds. Therefore VM apps developed using this approach can be deployed today over services such as Amazon EC2.

(**3**) *Capability.* As with the previous design option, the XenBlanket-based approach can easily support standalone, service and cooperative VM apps. VM apps cannot be composed easily, and a bundled VM app can best be created by composing the functionality of user-level applications within the nested management VM.

4.4 Hybrid Designs

Finally, it may be possible to create a hybrid design for a VM app that combines the designs discussed above. For example, suppose that a cloud provider decides to incorporate modifications to allow clients to plumb their I/O paths, in effect allowing clients to place middlebox VM apps between their work VMs and the platform's management VM. However, the cloud provider may be reluctant to include VMM modifications that allow VM apps to map memory pages of client VMs. In such cases, it is still possible to realize VM apps such as rootkit detectors using nested virtualization using either the second or the third design options.

5. CONCLUSIONS AND FUTURE WORK

Pre-configured software and technical services in the form of cloud apps, available via cloud markets, are surging. In this paper, we argued that the model adopted by existing cloud market is restrictive, and proposed a richer model for cloud marketplaces. We envisioned a cloud app market that supports the creation of standard utilities such as firewalls, NIDS, storage encryption, and VMI-based security tools, and allows clients to link them with their work VMs. To this end, we categorized potential cloud apps into four categories and described the key requirements to realize them in today's public cloud environments. We then explored the design space to actually implement our vision of rich VM apps. In the future, we plan to implement the prototype of the proposed app model, deploy it in the cloud, and evaluate its effectiveness.

Acknowledgments. This work was funded in part by NSF grants CNS-0831268, CNS-0915394, and CNS-0952128.

REFERENCES

[1] Amazon Machine Images (AMIs). http://aws.amazon.com/amis.
[2] LibVMI. http://code.google.com/p/vmitools/.
[3] MiniOS. http://sourceforge.net/projects/minios/.
[4] The Cloud Market. http://thecloudmarket.com.
[5] TinyLinux. http://tiny.seul.org/en/.
[6] M. Ben-Yahuda, M. D. Day, Z. Dubitsky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B. Yassour. The Turtles project: Design and implementation of nested virtualization. In *OSDI*, 2010.
[7] Sven Bugiel, Stefan Nurnberger, Thomas Poppelmann, Ahmad-Reza Sadeghi, and Thomas Schneider. Amazonia: When elasticity snaps back. In *ACM CCS*, 2011.
[8] S. Butt, A. Lagar-Cavilla, A. Srivastava, and V. Ganapathy. Self-service cloud computing. In *ACM CCS*, 2012.
[9] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live Migration of Virtual Machines. In *USENIX NSDI*, 2005.
[10] T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *NDSS*, 2003.
[11] B. Kauer, P. Verissimo, and A. Bessani. Recursive virtual machines for advanced security mechanisms. In *DCDV*, 2011.
[12] L. Litty, H. A. Lagar-Cavilla, and D. Lie. Hypervisor Support for Identifying Covertly Executing Binaries. In *USENIX Security*, 2008.
[13] Bryan D. Payne, Martim Carbone, and Wenke Lee. Secure and flexible monitoring of virtual machines. In *ACSAC*, December 2007.
[14] A. Srivastava and J. Giffin. Tamper-resistant, application-aware blocking of malicious network connections. In *RAID*, Sep 2008.
[15] A. Srivastava, Himanshu Raj, J. Giffin, and Paul England. Trusted VM snapshots in untrusted cloud infrastructures. In *RAID*, 2012.
[16] Jinpeng Wei, Xiaolan Zhang, Glenn Ammons, Vasanth Bala, and Peng Ning. Managing Security of Virtual Machine Images in a Cloud Environment. In *CCSW*, 2009.
[17] D. Williams, H. Jamjoom, and H. Weatherspoon. The Xen-Blanket: Virtualize Once, Run Everywhere. In *ACM EuroSys*, Apr 2012.