Unshackle the Cloud!

Dan Williams[†], Eslam Elnikety[‡], Mohamed Eldehiry[‡],
Hani Jamjoom[*], Hai Huang[*], and Hakim Weatherspoon[†]

[†]Cornell University, Ithaca, NY
[*]IBM T. J. Watson Research Center, Hawthorne, NY
[‡]King Abdullah University of Science and Technology, Thuwal, Saudi Arabia

Abstract

Infrastructure-as-a-Service (IaaS) clouds are evolving from offering simple on-demand resources to providing diverse sets of tightly-coupled monolithic services. Like OS kernels of the 1980's and 1990's, these monolithic offerings, albeit rich in features, are significantly constraining users' freedom and control over the underlying—cloud—resources. For example, we are unaware of a true hybrid cloud, where its users can migrate virtual machines freely across clouds. This paper argues for a new type of IaaS cloud, an *xCloud*, that builds on ideas from extensible OSs to give users the flexibility to install custom cloud extensions, which can address the limitations outlined above. We describe the design space for xClouds, including a practical approach for transforming today's public clouds into xClouds.

1 Introduction

> "*Nature is a mutable cloud which is always and never the same.*"
>
> —Ralph Waldo Emerson

Whereas Infrastructure-as-a-Service (IaaS) clouds once provided a simple bare-bones virtual machine (VM) abstraction, they are now evolving into increasingly diverse, feature-rich offerings. On the surface, this is advantageous: cloud users on Amazon EC2, for example, enjoy tools such as CloudWatch (integrated monitoring), AutoScaling, and Elastic Load Balancing. Beneath the surface, however, users are constrained: cloud provider features are rapidly becoming synonymous with vendor lock-in [1, 2], which is a symptom of a larger problem. Users are completely dependent on the provider for any hypervisor-level features. Tools and techniques at the hypervisor-level—enabling increased portability, availability [9], security [10], efficiency [24] and performance [14]—are impossible for users to implement themselves.

The current state of clouds resembles a point in the evolution of operating system (OS) kernels. In particular, extensible systems (such as exokernels [11], SPIN [5], and VINO [19]) emerged to solve certain limitations in monolithic kernels. These systems were motivated, in part, by applications' inability to define their own tailored hardware abstractions, just as applications on today's clouds are unable define their own virtual and physical hardware abstractions.

In this position paper, we argue that the abstraction of an extensible cloud, or *xCloud*, is essential for certain cloud applications. At the same time, we point out that the deployment of an xCloud must not depend on support from cloud providers. For example, current calls for standardization across multiple cloud providers will likely take years to implement, if ever. Cross-provider live migration—an important and commonly sought out feature—continues to remain a fantasy. On the other hand, the user with the ability to customize and homogenize the cloud will be able to implement cross-platform live migration immediately.

2 What's Wrong with Today's Clouds?

The uses of today's clouds are extensive: users range from a person deploying a single VM to an entire IT department or enterprise deploying hundreds or thousands of VMs. A rich array of services, third-party cloud management tools [8], and middleware [6, 16] operate at the VM-level to provide useful high-level functionality to cloud users. However, beneath this superficial VM-level veneer, the deployment of efficient, portable, innovative applications—especially by large cloud users attempting to efficiently manage a hybrid cloud—is being hindered by two main shortcomings:

Immutable Hypervisors: The hypervisor, or virtual machine monitor (VMM), in today's clouds is controlled by the provider, leaving users with little or no say as to

Abstraction Level	Feature
Today's Clouds	Application monitoring Auto-scaling Non-live migration
Mutable Hypervisor	Page sharing [12,21] Overdriver [24] Revirt [10] Remus [9] Live migration [7] Cross-provider live migration
Exposed Hardware	vSnoop [14] Superpages [18] Page coloring [15] Non fate-sharing Unsupported paravirtualization

Table 1: Cloud abstractions and extensions they enable

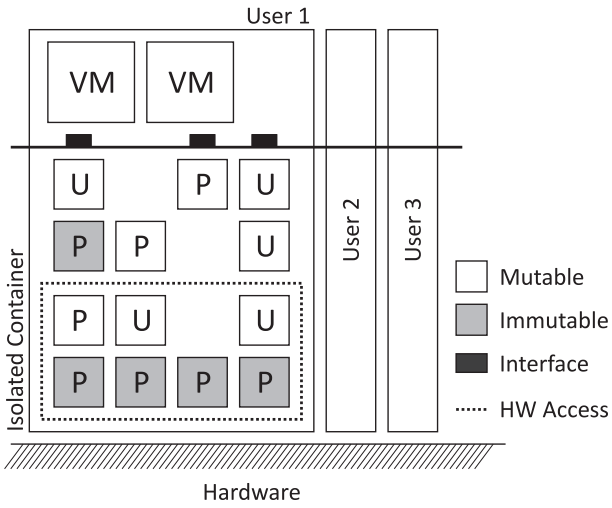


Figure 1: General extensibility architecture. **U** and **P** denote user and provider installed modules, respectively.

what hypervisor-level functionality is implemented or exposed. For example, no cloud currently exists with a hypervisor that allows users to maximally utilize their leased VMs through techniques like page sharing [12,21] or aggressive oversubscription [24]. Live VM migration [7] between multiple clouds—public or private—is virtually impossible. Innovative hypervisor-level techniques for high availability [9] or intrusion detection [10] are unavailable, while further customization and experimentation at this level is stifled.

Buried Hardware: Today's clouds bury the details of hardware beneath a virtual machine abstraction. Users must depend on the provider to expose everything from efficient I/O interfaces to physical fate-sharing information. Moreover, they cannot implement hardware-dependent tricks to squeeze the best performance out of the rented resources. Time-sensitive tasks, such as TCP acknowledgment [14], are difficult. Superpage [18] utilization is not efficient on virtual memory that may not be contiguous, and performance opportunities like page coloring [15] are also lost.

Table 1 provides examples of cloud extensions, some of which are available today. However, there is a large set of features—spanning performance, security, and portability—that require a mutable hypervisor. A further set of performance-related features require control or visibility at the hardware level. It is important to note that despite advocating for mutability, we believe that IaaS clouds should continue to provide a VM abstraction. This approach is fundamentally different from that of cloud operating systems that expose an OS abstraction [22].

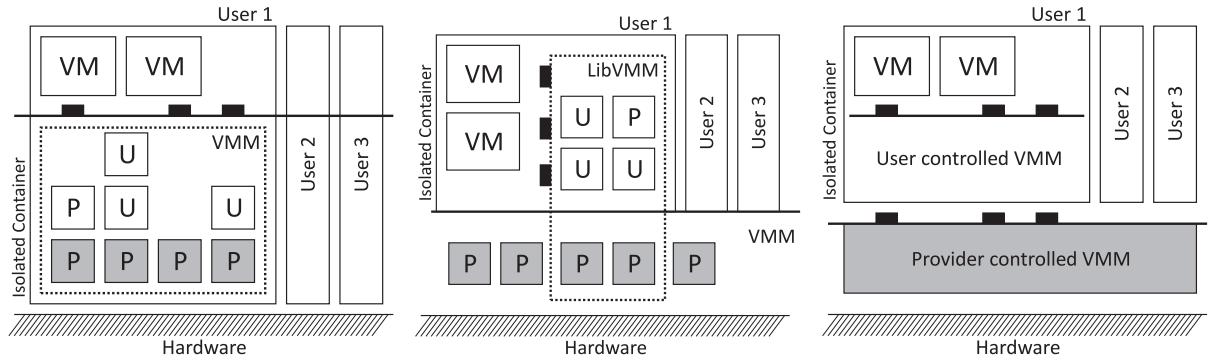
3 xClouds

In response to today's clouds, made up of immutable hypervisors and buried hardware, we propose an xCloud. Its general components are shown in Figure 1. Like today's IaaS clouds, an xCloud ultimately exposes a VM-like interface, upon which cloud users can run VMs. Unlike today's clouds, the VM-like interface can be customized and user-defined hypervisor-level functionality can be introduced to directly interface with the hardware. Thus, an xCloud exposes both a mutable hypervisor and the underlying hardware.

We consider the hypervisor to be made up of a number of modules that interact to comprise the inner-workings of the cloud provider. The provider will likely implement modules that multiplex hardware and enforce protection, such as the isolated containers in Figure 1 that protect cloud users from one another. Modules that implement functionality essential to the operation of the cloud, such as protection and accounting, are immutable. Some modules may be modified by users; others, implementing innovative or experimental interfaces, may be completely supplied by the cloud user. Modules depicted within a dotted box require access to the hardware.

There are a number of design alternatives for arranging the components in Figure 1. These alternatives, shown in Figure 2, are largely inspired by seminal work on extensible kernels.

Download Extensions into the VMM: SPIN [5] and VINO [19] are two systems from the 1990's in which extensions, or grafts, can be downloaded into the kernel, and run safely. Safety is provided mostly at the language level, using techniques like safe languages (Modula-3) and software fault isolation. An xCloud architecture that



(a) Download extension modules into VMM (b) Export hardware through VMM (c) Add another VMM

Figure 2: Three design alternatives for xClouds. The shading scheme is identical to Figure 1.

adopts this design is shown in Figure 2(a). The hypervisor becomes mutable by allowing user-defined or user-modified modules to be downloaded into the kernel. Similar to the extensible OSs, this must be done safely, such that other modules, especially immutable provider modules, are protected. Since modules are executing in the hypervisor with privilege, they can be granted direct access to the hardware.

Expose Hardware through the VMM: Exokernel [11] is a system that achieves extensibility by exposing hardware directly to applications, to the extent that the actual hardware names and addresses are visible to applications. Management of the hardware, traditionally done by the OS kernel, is performed by a library OS (libOS) that can be completely custom built and linked into the application. The kernel, on the other hand, only enforces protection between applications, which can be complex [13]. With the renewed interest in virtualization in the early 2000's, paravirtualization revisited many of these ideas, with the Denali isolation kernel [23] exposing much of the hardware, and implementing the traditional OS as a library, linked into the application. Xen [3] also adopted a paravirtualization approach and argued that full virtualization is not desirable when a guest OS needs to see real physical resources. Figure 2(b) shows a design in which the cloud provider exposes hardware to a "libVMM" under the control of a cloud user—analogous to a libOS on top of an Exokernel. The hardware is not buried, but exposed to the libVMM which is completely mutable.

Add Another VMM: Interest in nested virtualization is increasing as virtualization becomes ever more ubiquitous. Furthermore, nested virtualization has been shown to perform well, for example, the Turtles Project [4] has achieved performance within 6-8% of single-level virtualization for some workloads. Figure 2(c) shows how nested virtualization can be leveraged for extensibility in the cloud. When a user leases a VM instance, it installs a mutable, second layer hypervisor to run on top of the cloud provider's VM abstraction. The providers' and users' modules are implemented in the first and second layer hypervisors, respectively. It should be noted that existing techniques require modifications to the bottom-most hypervisor in order to expose virtualization hardware extensions to VMs.

Nested virtualization fundamentally differs from the previous two design alternatives. On the one hand, using nested virtualization, the hardware remains buried under the virtual machine abstraction, preventing a cloud user from implementing the class of performance-enhancements described in Section 2. On the other hand, nested virtualization has enormous potential for immediate and incremental deployment, even without provider cooperation. As an alternative to existing nested virtualization systems that require lower-layer VMM modifications, techniques such as paravirtualization can be applied inside a standard fully virtualized or even paravirtualized guest VM. As such, this mutable, second layer hypervisor implements an xCloud that can be immediately deployed on today's cloud platforms without requiring their involvement. These unexplored and unevaluated techniques for nested virtualization offer a compelling xCloud architecture, but will such an xCloud perform?

4 Will a Deployable xCloud Perform?

As discussed above, the nested virtualization approach has the advantage of rapid and immediate deployment on today's clouds. In this section, we show that nested virtualization can achieve reasonable performance without provider involvement. Then, we briefly discuss opportunities to achieve higher performance, still without any changes to the underlying platform.

For these experiments, we assume standard, hardware-assisted hypervisors that run at the lowest layer, similar to the HVM instances available from Amazon EC2.

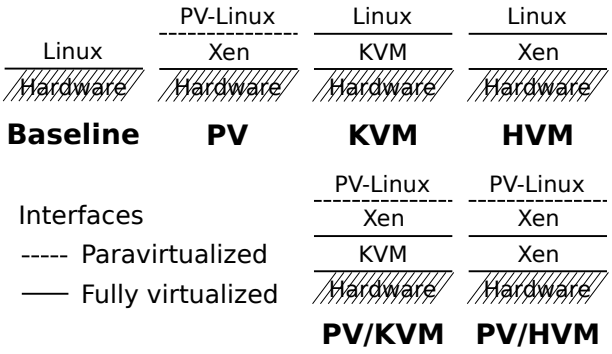


Figure 3: Virtualization configurations

		Single			Nested	
	Baseline	PV	HVM	KVM	PV / HVM	PV / KVM
double div (ns)	7.19	7.55	7.61	7.41	7.57	7.35
null call (μs)	.19	.37	.21	.20	.37	.38
fork proc (μs)	65.17	249.70	78.89	86.52	280.39	336.93

Table 2: Microbenchmarks using `lmbench`

Without hypervisor modifications to virtualize the processor features supporting virtualization [4], we must use other techniques to implement the second layer hypervisor, such as paravirtualization (e.g., Xen [3]), binary translation (e.g., VMWare [20]), or full emulation (e.g., QEMU).[1] Our experiments focus on using paravirtualized Xen,[2] and were performed on machines with 24 GB of RAM and dual 6-core 2.93 GHz Intel Xeon X5670 processors, which include advanced virtualization features like extended page tables. The virtualization configurations that we compare are described in Figure 3.

Table 2 shows the results of some `lmbench` microbenchmarks over the various setups (Figure 3). As expected, all arithmetic operations, like double division (shown), are minimally affected by virtualization. Simple operations like null system calls, are achieved in fully virtualized configurations without hypervisor support, and thus have matching performance. Paravirtualization, on the other hand, invokes the hypervisor on the system call, which must be bounced back up to the guest OS. Nesting does not introduce any extra overhead beyond that of PV. However, in nested environments, process fork generates 12-30% additional overhead over PV by inducing traps into the lowest layer hypervisor.

I/O typically stresses a virtualized system because of the inefficiencies of emulation required to fully virtualize I/O devices and handle interrupts. To determine the overhead introduced by the second layer hypervisor, we

[1]Results from experimentation with QEMU are not shown due to the abysmal performance of full emulation, which was up to two orders of magnitude slower.

[2]While Xen paravirtualization somewhat limits compatibility, we note that it is very popular; for instance, a large fraction of Amazon EC2's offerings are indeed paravirtualized.

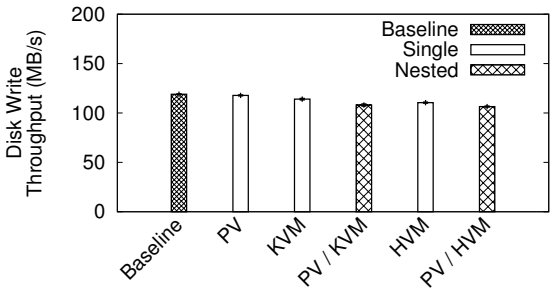


Figure 4: Disk throughput using `dd`

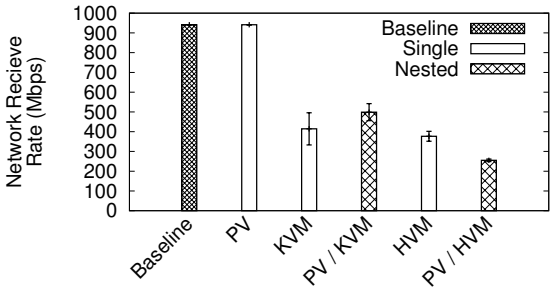


Figure 5: Network receive throughput using `netperf`

conducted experiments with disk I/O and network I/O. In the first experiment, we measure the performance of disk I/O by writing 1.6 GB of data to a disk partition with `dd` using blocks of size 256 K. Each result is the average of 5 trials. Figure 4 shows the throughput of the disk. We find that, for disk I/O, nested virtualization does not cause significant overhead, achieving 90% of native throughput.

In the second experiment, we measure the performance of network I/O. The network device typically generates more interrupts and requires more OS interaction than a disk, making it a more stressful test for virtualized environments. We ran `netperf` in each setup in order to determine how fast a guest could receive TCP network traffic generated from another machine across a 1 Gbps network. `netperf` was configured to use the default settings and a message size of 16,384 bytes. Each result is the average of 10 trials. Figure 5 shows the results. Considering a single layer of virtualization, it is immediately obvious that full virtualization (HVM, KVM) incurs a dramatic performance hit. HVM performs particularly poorly, achieving only 40% of the baseline throughput.[3] PV, which bypasses device emulation by using paravirtualization, achieves performance matching the baseline. This suggests that the poor performance of the nested setups is largely due to the device emulation of the first layer hypervisor. For instance, whereas the single layer of KVM virtualization reduced throughput by 47%, the second layer only reduces it by a further 9%.

[3]This is a known limitation of network virtualization in Xen due to inefficient I/O remapping [17].

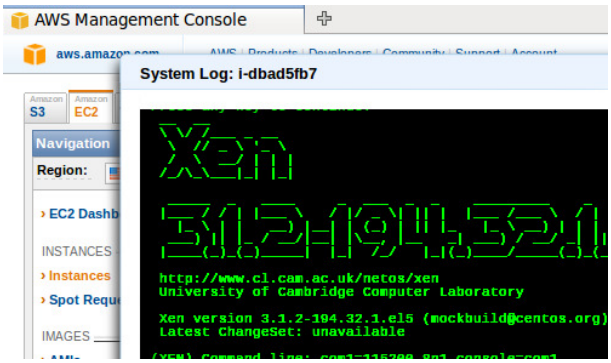


Figure 6: Screenshot of Xen booting on Amazon EC2

The performance impact of a second layer of virtualization appears to be a reasonable price to pay for a readily deployable xCloud. Paravirtualization is extremely important for device I/O, suggesting that incorporating paravirtualization through both layers may lead to even better performance.

5 Conclusion and Future Work

Cloud providers are trending towards complexity, diversity, and vendor lock-in. This trend is also limiting users' abilities to implement sophisticated, custom applications that require VMM- and HW-level control. This paper argues the need to address today's limitations through extensibility.

Creating an xCloud on today's infrastructure is within reach using nested virtualization as an extensibility layer. In preliminary experiments, we showed that its overhead is within acceptable limits given its relative ease of deployment. We also noted the potential performance benefits of using paravirtualized I/O drivers on both virtualization layers, eliminating the bottleneck presented by device emulation. In ongoing work, we have implemented *nested paravirtual* device drivers, that can transmit and recieve packets from the network at line speed on a Gigabit link. Furthermore, nested paravirtual drivers are required for compatibility with clouds that expose a paravirtual network interface, such as Amazon EC2. As shown in Figure 6, we have successfully run Xen on EC2 and are currently evaluating the performance of EC2 as an xCloud. Finally, we intend to implement nested paravirtualization on a fully paravirtualized EC2 instance, extending both the deployability and performance of xCloud. The xCloud webpage is available at `http://xcloud.cs.cornell.edu`.

Acknowledgments This work was performed while the first 3 authors were interns at the IBM T. J. Research Center in Hawthorne, NY. Also, the Cornell group is partially funded and supported by an IBM Faculty Award received by Hakim Weatherspoon and NSF TRUST.

References

[1] Abu-Libdeh, H., Princehouse, L., and Weatherspoon, H. RACS: a case for cloud storage diversity. In *Proc. of ACM SoCC* (Indianapolis, IN, June 2010).

[2] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. A view of cloud computing. *Communications of the ACM 53*, 4 (2010), 50–58.

[3] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the art of virtualization. In *Proc. of ACM SOSP* (Bolton Landing, NY, Oct. 2003).

[4] Ben-Yehuda, M., Day, M. D., Dubitzky, Z., Factor, M., Har'El, N., Gordon, A., Liguori, A., Wasserman, O., and Yassour, B.-A. The turtles project: Design and implementation of nested virtualization. In *Proc. of USENIX OSDI* (Vancouver, BC, Canada, Oct. 2010).

[5] Bershad, B. N., Savage, S., Pardyak, P., Sirer, E. G., Fiuczynski, M. E., Becker, D., Chambers, C., and Eggers, S. Extensibility, safety and performance in the SPIN operating system. In *Proc. of ACM SOSP* (Copper Mountain, CO, Dec. 1995).

[6] Campbell, R., Gupta, I., Heath, M., Ko, S. Y., Kozuch, M., Kunze, M., Kwan, T., Lai, K., Lee, H. Y., Lyons, M., Milojicic, D., O'Hallaron, D., and Soh, Y. C. Open cirrus^{TM} cloud computing testbed: federated data centers for open source systems and services research. In *Proc. of USENIX HotCloud* (San Diego, CA, June 2009).

[7] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. Live migration of virtual machines. In *Proc. of USENIX NSDI* (Boston, MA, May 2005).

[8] Clark, T. Rightscale. http://www.rightscale.com, 2010.

[9] Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N., and Warfield, A. Remus: high availability via asynchronous virtual machine replication. In *Proc. of USENIX NSDI* (San Francisco, CA, Apr. 2008).

[10] Dunlap, G. W., King, S. T., Cinar, S., Basrai, M. A., and Chen, P. M. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. In *Proc. of USENIX OSDI* (Boston, MA, Dec. 2002).

[11] Engler, D. R., Kaashoek, M. F., and O'Toole, J. W. Exokernel: An operating system architecture for application-level resource management. In *Proc. of ACM SOSP* (Copper Mountain, CO, Dec. 1995).

[12] Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A. C., Varghese, G., Voelker, G. M., and Vahdat, A. Difference engine: Harnessing memory redundancy in virtual machines. In *Proc. of USENIX OSDI* (San Diego, CA, Dec. 2008).

[13] Kaashoek, M. F., Engler, D. R., Ganger, G. R., Briceño, H. M., Hunt, R., Mazières, D., Pinckney, T., Grimm, R., Jannotti, J., and Mackenzie, K. Application performance and flexibility on exokernel systems. In *Proc. of ACM SOSP* (St. Malo, France, Oct. 1997).

[14] Kangarlou, A., Gamage, S., Kompella, R. R., and Xu, D. vSnoop: Improving TCP throughput in virtualized environments via acknowledgement offload. In *Proc. of ACM/IEEE SC* (New Orleans, LA, Nov. 2010).

[15] Kessler, R. E., and Hill, M. D. Page placement algorithms for large real-indexed caches. *ACM Transactions on Computer Systems 10*, 4 (Nov. 1992), 338–359.

[16] Maximilien, E. M., Ranabahu, A., Engehausen, R., and Anderson, L. C. IBM altocumulus: a cross-cloud middleware and platform. In *Proc. of ACM OOPSLA Conf.* (Orlando, FL, Oct. 2009).

[17] Menon, A., and Zwaenepoel, W. Optimizing network virtualization in Xen. In *USENIX Annual Technical Conference* (Boston, 2006).

[18] Navarro, J., Iyer, S., Druschel, P., and Cox, A. Practical, transparent operating system support for superpages. In *Proc. of USENIX OSDI* (Boston, MA, Dec. 2002).

[19] Seltzer, M. I., Endo, Y., Small, C., and Smith, K. A. Dealing with disaster: Surviving misbehaved kernel extensions. In *Proc. of USENIX OSDI* (Seattle, WA, Oct. 1996).

[20] Sugerman, J., Venkitachalam, G., and Lim, B.-H. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In *Proc. of USENIX Annual Technical Conf.* (Boston, MA, June 2001).

[21] Waldspurger, C. A. Memory resource management in VMware ESX server. In *Proc. of USENIX OSDI* (Boston, MA, Dec. 2002).

[22] Wentzlaff, D., Gruenwald, III, C., Beckmann, N., Modzelewski, K., Belay, A., Youseff, L., Miller, J., and Agarwal, A. An operating system for multicore and clouds: mechanisms and implementation. In *Proc. of ACM SoCC* (Indianapolis, IN, June 2010).

[23] Whitaker, A., Shaw, M., and Gribble, S. D. Scale and performance in the Denali isolation kernel. In *Proc. of USENIX OSDI* (Boston, MA, Dec. 2002).

[24] Williams, D., Jamjoom, H., Liu, Y.-H., and Weatherspoon, H. Overdriver: Handling memory overload in an oversubscribed cloud. In *Proc. of ACM VEE* (Newport Beach, CA, Mar. 2011).