Research Statement

Dan Williams

My research interests generally revolve around building systems, especially the development of cloud software, hypervisors, and operating system kernels. My Ph.D. thesis research at Cornell focuses on how to efficiently deploy large systems on clouds irrespective of whether they are public, private or a mixture of both. Also at Cornell, I researched operating system mechanisms to support trustworthy computing; in particular, how to reduce the trusted computing base (TCB). My general approach to addressing systems research challenges is based on solid understanding through implementation and evaluation of real systems.

1. Ongoing Research: Large Cloud Deployments

While Infrastructure as a Service (IaaS) clouds have emerged as a convenient method for leasing virtual machines (VMs), deploying large enterprise-grade applications remains difficult and highly inefficient. For example, an IT department that manages hundreds or thousands of VMs faces significant challenges in migrating to the cloud, including:

- *Under-utilization of Leased Resources:* At scale, the common practice of provisioning VMs to handle peak loads, or overprovisioning, begins to result in significant under-utilization of resources.
- *Inflexibility of Cloud Services:* Large VM deployments often require VM management tools or deployment on multiple—public or private—cloud providers.

I have designed, implemented, and evaluated two systems that address these challenges: Overdriver and the Xen-Blanket, respectively. I am also interested in pursuing future research projects inspired by the insights gleaned from each.

1.1 Overdriver

In large VM deployments, the traditional conservative strategy of overprovisioning each VM with enough physical resources to support relatively rare peak load conditions leads to significant under-utilization of resources most of the time. Oversubscription promises to increase utilization at a cost: an increased likelihood of *overload*, or a condition in which one or more VMs do not have the resources to complete a task without encountering performance degradation.

Focusing on the mitigation of *memory* overload because of its severe effects on performance, I designed, implemented and evaluated a system called Overdriver [9]. The key insight behind Overdriver is an observation, from my own analysis of current data center logs and realistic Web workloads, that memory overload is largely transient: up to 88.1% of overloads last for less than 2 minutes. This observed *overload continuum*, containing not only sustained but also transient overload, suggests that no one mitigation approach will be optimal for all overloads. In particular, heavyweight techniques, like VM migration, are better suited to sustained overloads, whereas lightweight approaches, like network memory, are better suited to transient overloads.

Overdriver adaptively takes advantage of these tradeoffs to manage a full continuum of sustained and transient overloads. On transient overloads, rather than swapping memory pages to disk, overloaded VM's memory pages are written to (and read from) network memory. A threshold-based mechanism actively monitors the duration of overload in order to decide when an overload should be classified as sustained and a VM migration operation should occur. The thresholds are adjusted based on VM-specific probability overload profiles, which Overdriver learns dynamically. Overdriver's adaptation between network memory and VM migration reduces potential application performance degradation, while ensuring the chance (and overhead) of unnecessary migration operations remains low.

This research has produced important insights regarding the amount and distribution of excess resources that must be reserved to handle overload events. In particular, given a set of applications and their overload characteristics, an optimal degree of oversubscription exists. Applying concepts from Overdriver, large VM deployments can be safely oversubscribed in terms of memory, and the utilization of leased resources can be increased.

1.2 The Xen-Blanket

Large VM deployments require flexibility with regard to the underlying cloud infrastructure. Many common and experimental tools for managing large VM deployments, including live VM migration [5], page sharing [14], and Overdriver, rely

on hypervisor-level techniques not exposed to cloud users in today's clouds. Provider support for multi-cloud deployments is extremely limited if they exist at all.

In my research, I have proposed the concept of an extensible cloud [8] as fundamental for migrating large VM deployments to the cloud. I have also fully developed a prototype extensible cloud called the Xen-Blanket [10]. The Xen-Blanket is a system that leverages nested virtualization to transform existing heterogeneous clouds into a uniform *user-centric* homogeneous offering without requiring special support from underlying cloud providers. I have deployed the Xen-Blanket on both Xen-based and KVM-based hypervisors, on public and private infrastructures within Amazon EC2, IBM, and Cornell University.

The Xen-Blanket leverages nested virtualization to run a second-layer hypervisor as a guest on top of a variety of public or private clouds, forming a *Blanket layer*. The Blanket layer exposes a homogeneous interface to second-layer guest VMs and is completely customizable. Hypervisor-level techniques and management tools, like those described above, can all be implemented inside the Blanket layer. Unlike existing nested virtualization techniques (like the Turtles project [3]), which require changes to the providers' hypervisors, the Blanket layer contains *Blanket drivers* that allow it to run on heterogeneous clouds while hiding interface details of the underlying clouds from guests.

The Xen-Blanket enables true multi-cloud deployment. For example, I have live migrated VMs to and from Amazon EC2 with no modifications to the VMs. Furthermore, the user-centric design of the Xen-Blanket affords users the flexibility to oversubscribe resources such as network, memory, and disk. As a direct result, a Xen-Blanket image on EC2 can host 40 CPU-intensive VMs for 47% of the price per hour of 40 small instances with matching performance.

1.3 Future Directions

With the insights and infrastructure produced from the Overdriver and Xen-Blanket projects, there are many interesting research challenges surrounding how to exploit multiple clouds in an efficient cloud abstraction; in other words, how to create efficient clouds within clouds. For example, I plan to investigate the implications of a full cloud software stack, like Eucalyptus [11] or OpenStack [1], implemented on the Xen-Blanket across multiple cloud providers. In the short term, I plan to further reduce overprovisioning for memory in clouds and address network incompatibilities that arise in a multi-cloud environment.

Memory Clouds Overprovisioning of memory occurs because of difficulties in predicting the memory usage of VMs. Truly elastic resources, such as storage, do not require prediction: an abstraction of infinite storage is presented, while a VM pays for what it actually uses. I am interested in pursuing the idea of truly elastic memory or a *memory cloud*, in which elastic memory behaves like elastic storage. With a memory cloud, VMs see a virtually unlimited amount of memory and pay for only what they use. The underlying hypervisor-level software must maintain the illusion of infinite memory without VMs perceiving that some of the memory may not be local. Techniques like those explored in Overdriver will be essential, and emerging architectures with extremely fast inter-node communication, such as BlueGene, may facilitate the implementation of an efficient memory cloud abstraction.

Virtual Networking in Clouds The Xen-Blanket grants hypervisor-level control over computation, memory and storage resources in today's clouds. However, this abstraction is insufficient for large deployments that require control over the network. For example, VMs require a specific IP address, or VLANs may be required to isolate certain types of traffic. I am in the process of building and evaluating a system that creates a virtual network within the Xen-Blanket layer, invisible to VMs. Every VM running on the Blanket layer is connected to a virtual switch in the Xen-Blanket. Virtual switches are connected to each other with layer 2 tunnels that form an overlay network that may span multiple clouds. Therefore, all VMs on the virtual network can communicate with each other with layer 2 protocols while maintaining their network configuration. This affords cloud deployments unprecedented flexibility: VMs on different clouds can share a VLAN; they can live migrate from one cloud to another; and new or emerging network features like OpenFlow can be supported on the virtual network even if no physical switches support it. Ongoing work is examining how to create a scalable overlay to support the virtual network.

2. Prior Research: The Nexus Operating System

The Nexus [13] is a new operating system designed to support trustworthy computing. I was involved in building the Nexus from scratch, making significant code contributions in every major kernel subsystem including memory management, thread management, interrupt delivery, device support, and storage. My research focused on ensuring that the Nexus was designed to be trustworthy, in particular, by maintaining a small trusted computing base (TCB). My research contributions shrink the TCB of traditional operating systems in two areas.

First, the storage system of traditional operating systems is in the TCB. In particular, the disk contents must be trusted because the OS cannot even detect a simple attack in which a disk image is duplicated and replayed while the

machine is powered down. I designed, implemented, and evaluated a storage system to offer integrity and confidentiality protection—even across reboots—and eliminate the need to trust disk contents. It contains an abstraction called Secure Storage Regions (SSRs) that leverage limited Trusted Platform Module (TPM) storage resources to provide integrity- and confidentiality- protected, replay-proof, persistent storage. In this research, I applied the technique of using Merkle hash trees [12] as a low-cost mechanism to protect the integrity of data. I also investigated the optimal selection of blocksize for Merkle hash trees [6] and derived an analytical model that describes the cost of incremental updates to a Merkle hash tree given the total size of a memory region to be protected and the number of modified memory locations in each transaction. Using this model, the SSR subsystem can numerically determine the blocksize that minimizes the cost of performing updates to the tree.

Second, device drivers constitute over half of the source code of many operating system kernels, with a bug rate up to seven times higher than other kernel code [4], yet typically run in the kernel with supervisor privilege. Even in user space, device drivers execute hardware I/O operations and handle interrupts, allowing them to cause device behavior that compromises the integrity or availability of the kernel or other programs. I designed, implemented, and evaluated a user-space driver subsystem that removes drivers from the TCB by running them without supervisor privileges and constrains their interactions with hardware devices [7]. The key component that I built in the device subsystem is a trusted reference validation mechanism (RVM) [2] that mediates all interaction between device drivers and devices. The RVM invokes a device-specific reference monitor to validate interactions between a driver and its associated device, thereby ensuring the driver conforms to a device safety specification (DSS), which defines allowed and, by extension, prohibited behaviors. My research also produced a domain-specific language to express the DSS, which defines a state machine that accepts permissible transitions by a monitored device driver, and a DSS compiler to translate each DSS into a reference monitor that implements the state machine. Every operation by the device driver is vetted by the reference monitor, and operations that would cause an illegal transition are blocked. By fundamentally reducing the TCB, the Nexus architecture provides insight into building trustworthy systems.

References

[1] OpenStack. `http://www.openstack.org/`, Oct. 2010.

[2] J. P. Anderson. Computer security technology planning study—Vol. II. Technical Report ESD-TR-73-51 Vol. II, Electronic Systems Division, AFSC, L. G. Hanscom Field, Bedford, MA, Sept. 1972.

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

[4] A. Chou, J.-F. Yang, B. Chelf, S. Hallem, and D. Engler. An empirical study of operating system errors. In *Proc. of ACM SOSP*, Banff, Canada, Oct. 2001.

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

[6] **D. Williams** and E. G. Sirer. Optimal parameter selection for efficient memory integrity verification using merkle hash trees. In *Proc. of the IEEE NCA Trustworthy Network Computing Workshop*, Cambridge, MA, Aug. 2004.

[7] **D. Williams**, P. Reynolds, K. Walsh, E. G. Sirer, and F. B. Schneider. Device driver safety through a reference validation mechanism. In *Proc. of USENIX OSDI*, San Diego, CA, Dec. 2008.

[8] **D. Williams**, E. Elnikety, M. Eldehiry, H. Jamjoom, H. Huang, and H. Weatherspoon. Unshackle the cloud! In *Proc. of USENIX HotCloud*, Portland, OR, June 2011.

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

[10] **D. Williams**, H. Jamjoom, and H. Weatherspoon. The Xen-Blanket: Virtualize once, run everywhere. In *Submission*, Oct. 2011.

[11] Eucalyptus Systems, Inc. Eucalyptus open-source cloud computing infrastructure - an overview. `http://www.eucalyptus.com/pdf/whitepapers/Eucalyptus_Overview.pdf`, Aug. 2009.

[12] R. C. Merkle. Protocols for public key cryptosystems. In *Proc. of IEEE Symposium on Security and Privacy*, Oakland, CA, Apr. 1980.

[13] E. G. Sirer, W. de Bruijn, P. Reynolds, A. Shieh, K. Walsh, **D. Williams**, and F. B. Schneider. Logical attestation: An authorization architecture for trustworthy computing. In *Proc. of ACM SOSP*, Cascais, Portugal, Oct. 2011.

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