Scale and Performance in the Denali Isolation Kernel

In contrast to some of the more popular virtualization solutions we have read about and discussed so far, the Denali isolation kernel is targeted at a fairly narrow set of problems: that of supporting Internet services running untrusted code on a very large scale. In addition, the use model within which this discussion takes place assumes the need (or desirability) for Internet services to be pushed into third-party hosting hardware. As the article makes clear, this allows “separating the deployment of services from the management of the physical infrastructure on which they run.” The big concerns in this environment are therefore security, performance, and scalability. The authors claim that more than 10,000 virtual machines can operate securely on one commodity hardware platform. In the process, the ability to support unmodified “legacy” operating systems is abandoned as a design goal.

The Denali isolation kernel runs as a thin layer directly on the hardware. The design goals of security, performance and scalability are realized by keeping everything as simple as possible and focusing design on the system's primary application. The kernel's main design principles are:

1. Expose low-level resources rather than high-level abstractions.
2. Prevent direct sharing by exposing only private, virtualized namespaces.
3. Zipf's Law implies the need for scale.
4. Modify the virtualized architecture for simplicity, scale, and performance.

By exposing low-level resources instead of high-level abstractions, security is greatly increased. This philosophy is in line with the security principle of economy of mechanism. Operating systems with complex APIs have become notorious for the vulnerabilities and complex layering invites “layer below” attacks, as the authors point out. One important manifestation of this approach is a tiny code base that is obviously going to be easier to debug and maintain.
The very nature of the application, running untrusted Internet services on a large scale, argues against the need for much sharing between VMs. Therefore, the only sharing allowed between VMs is via the virtual network. This principle increases security and greatly simplifies kernel design by eliminating the need for complex access control policies.

I'm taking the authors at their word that the popularity of Internet services follows a Zipf distribution. Clearly a number of arguments could be made for the ability of the isolation kernel to support scalability, for reasons of simple economy if nothing else.

Finally, by abandoning the idea that legacy and unmodified operating systems be supported, the Denali isolation kernel stands to benefit from greatly simplifying the virtual architecture for guest VMs, particularly on hardware that was not designed to support virtualization. This principle seems like a no-brainer, particularly after our discussions of the VMware papers. This is an area that obviously caused the VMware folks a great deal of frustration. In some cases, they developed innovative approaches to solve problems in a way that introduced some additional virtualization overhead. In other cases, the only solution was to live with virtualization bottlenecks in exchange for the flexibility of being able to run unmodified commodity operating systems as guests.

The Denali isolation kernel achieves its design principles in a number of ways, with simplicity being a primary tool throughout its implementation:

1. The virtual instruction set is a subset of the x86 instruction set. The problematic instructions that give other VMMs fits are simply not supported. This is possible because there is no attempt to run unmodified guest OSs.

2. The problem of idling guests wasting CPU cycles is solved rather neatly by implementing an “idle-with-timeout” instruction, in which idling guests actually halt for either a bounded time or until an interrupt arrives for that guest. In addition, CPU scheduling uses a simple, starvation free two-level policy. There's a gatekeeper policy that selects a subset of active VMs to be admitted, as long as there are memory pages available. The scheduler then uses a round-robin
scheduling algorithm for admitted VMs.

3. The memory architecture is greatly simplified, in comparison with other virtualization systems. A VM's address space has two components: a memory area that is accessible to the VM and a protected area that only the isolation kernel can use. There's also a statically allocated swap space assigned to each VM that is large enough to hold the entire VM-visible memory space.

4. The virtual IO devices are greatly simplified over actual devices or even the virtual devices used in most other virtualization systems. For example, in nearly all cases, sending and receiving network traffic requires more than 10 programmed IO instructions in each case. The Denali simplification reduces this to one PIO each for sending and receiving. Numerous other aspects of virtual IO are simplified and streamlined in order to reduce overhead.

Management of virtualization systems is accomplished through a designated supervisor VM. Much of the complexity that would normally be present in the OS kernel, such as a network stack, is present in the supervisor VM.

Overall, I find that this article makes a strong case for isolation kernels to address the problem of providing untrusted Internet services on a large scale. The idea of running more than 10,000 VMs on a single commodity hardware platform fairly boggles the mind. The market for this solution may be fairly narrow, but the availability of isolation kernels makes deployment of large-scale infrastructure seem very feasible and affordable.