JAKE WIRES AND ANDREW WARFIELD

beyond blocks and files

Jake Wires received his M.S. in computer science from the University of British Columbia. He currently works in the Datacenter and Cloud Division at Citrix, where his focus is storage virtualization.

Jake.Wires@Citrix.com

Andrew Warfield is an assistant professor in the Department of Computer Science at the University of British Columbia.

andy@cs.ubc.ca

VIRTUAL MACHINES (VMS) CHANGE HOW file and storage systems need to work. Most conventional file systems were designed with the assumption that files would be accessed only through the operating system's file interface. This assumption seemed innocuous when operating systems owned their hardware, but virtual machines use virtual disks owned by virtual machine monitors (VMMs)—and now VMMs want an interface to access VM files too. Presently, VMMs are mostly limited to operating at the block layer, but in order to efficiently provide features such as versioning and deduplication they need to operate at the file system layer. Moreover, the problem of managing large numbers of VMs would be greatly simplified if VMMs better understood files. New file systems designed for use in virtualized operating systems should expose a file interface to VMMs and should better express data dependencies so that files can be safely manipulated from outside VMs.

As fans of virtualization may already well know, too much convenience can be a burden. In an era where the proliferation of real, expensive hardware already frequently motivates "spring cleaning" mass emails from IT departments, the emerging ability of users to spawn virtual machines at their pleasure can lead to managerial headaches. For example, while a system administrator might be quite pleased when she first discovers how easy it is to create a thousand Windows XP VMs, her spirits may falter a bit after she finds that each VM must be customized with individual SIDs and AD credentials if it is to be very useful on the corporate LAN. And she may grow downright frustrated when, a few months after distributing all these shiny new VMs, she finds that every one of them needs to be upgraded—without disrupting any changes users might have made.

Current technologies offer appealing solutions for managing the storage consumed by VMs, but managing the data produced by VMs is still very much an open problem. In many ways, this is an issue of perspective: the advent of VMMs challenges the traditional view that a disk and its files belong primarily to an operating system. The more popular VMs become, the more important it will be to expose OS data to VMMs in meaningful ways.

The familiar debate between block and file-oriented interfaces is no less germane to VMs than physical hardware, although virtualization may add a few new twists. The block interface, as the argument goes, is sublime in its simplicity: it is stateless, straightforward, and OS-agnostic. The file system interface, on the other hand, is often more relevant: it defines much richer storage abstractions and is better aligned with the way users typically reason about their data. This relevance comes at a cost, though:

- File systems are complex and often intricately entwined with other components of the operating system, such as page caches and virtual memory managers.
- File systems must satisfy sophisticated consistency requirements along performance-critical data paths.
- File systems tend to exhibit much greater variation across operating systems.

In general, it is easier for VMMs to interpose on guest VMs at the block layer than at the file system layer. Essential features such as thin provisioning and fast cloning are simple to implement behind the block interface, where they can easily support legacy OSes. Additional features such as versioning and deduplication can be implemented at the block layer as well, although purists might offer arguments for moving these features into the file system. There is very little benefit, for instance, in versioning things like Windows page files and hibernation files—old versions of such files are, for all practical purposes, worthless—but when operating at the block level, it is very difficult to avoid doing so. The upshot in this case is that with block-level versioning, disk snapshots intended to preserve a few kilobytes of user data may end up wasting gigabytes of disk space.

But putting matters of expediency aside, these block-level technologies share a noteworthy characteristic: they all contribute to making a mess of the otherwise simple block layer. While cloning a virtual disk is almost free, merging diverged clones is nearly impossible. Copy-on-write disks provide a quick path to versioning, but they introduce cumbersome dependency chains. Deduplication can reclaim storage space, but it also effectively invalidates disks for use with any tools that don't understand the deduplicator metadata. It may be tempting to ignore these issues when one's main concern is ticking feature check-boxes, but as systems begin to see extended use in the real world, the growing accumulation of interdependent but divergent virtual disks can pose unwieldy problems.

If block-level implementations suffer from such drawbacks, why don't VMMs start plugging into file systems? One major obstacle is that, irrespective of all the hooks and probes and monitors we have thus far attached to VMs, file systems have remained black boxes, and efforts to expose their interfaces to the VMM seem to call for more of the pickax than the scalpel. Even if it is feasible to teach VMMs about the on-disk layout of file systems, this alone would not be enough to provide features such as versioning and deduplication, because of issues such as write ordering and cache consistency within the VM. Interposing on VM file systems is a major effort that would require OS-specific implementations, introduce considerable security risks, and likely require a great deal of maintenance over time as VM file systems grow and evolve.

Such challenges have led to the proposal of new storage abstractions such as object-based disks and file/block hybrids like "flocks" (*not* your standard mutex primitive—perhaps it's inevitable that we'll one day hear clamoring for the widespread adoption of "biles"). These abstractions offer some intriguing new properties. Imagine, for instance, that object-based storage had been adopted 10 or 20 years ago: VMMs would be well positioned to provide features like file-grained versioning and single-instance storage while still hiding behind an arguably tractable, OS-agnostic interface.

But what, after all, is in a name? That which we call a file, by any other name would be as complex. While most OS interfaces are designed to isolate system resources, file systems (and particularly file system namespaces) are peculiar in that they offer opportunities to introduce odd dependencies and circumvent isolation. With a bit of hand waving we can relegate the problem of files to object-based disks, and in so doing we can even congratulate ourselves a bit for better separating storage and namespace implementations, but ultimately we're left with containers of application-level information. If VMMs were able to manipulate these containers they could provide new features to a variety of OSes, but would we really be satisfied?

An especially prickly example here is VM upgrades. Administrators would like the ability to push OS and application updates down onto VM images without disturbing individual users' data. If VMMs recognized file objects, they could enforce read-only or copy-on-write policies for system-administered files, offering greater confidence that these files could be upgraded safely. But it seems doubtful that policies could be derived which would offer users the flexibility they demand while still guaranteeing that their personal customizations would be completely impervious to disruption, direct or otherwise, by system updates. In the end, no matter how transparent the structure of persistent data becomes, there will always be some amount of semantic information that will reside beyond the purview of administrators and limit their ability to safely manipulate VM disk images.

But maybe there are things we can do to mitigate these problems. For starters, the emerging presence of large VM deployments warrants a reevaluation of what a file is and who it is for. Perhaps we should even look beyond blocks and files to see if we can't find better ways of structuring VM semantics. Developing more effective methods of expressing data dependencies and enforcing isolation in the storage stack should be a high priority. As well, new standards of scalability are called for; just as current file systems allow us to manage thousands of files, new storage environments should let us manage thousands of file systems.