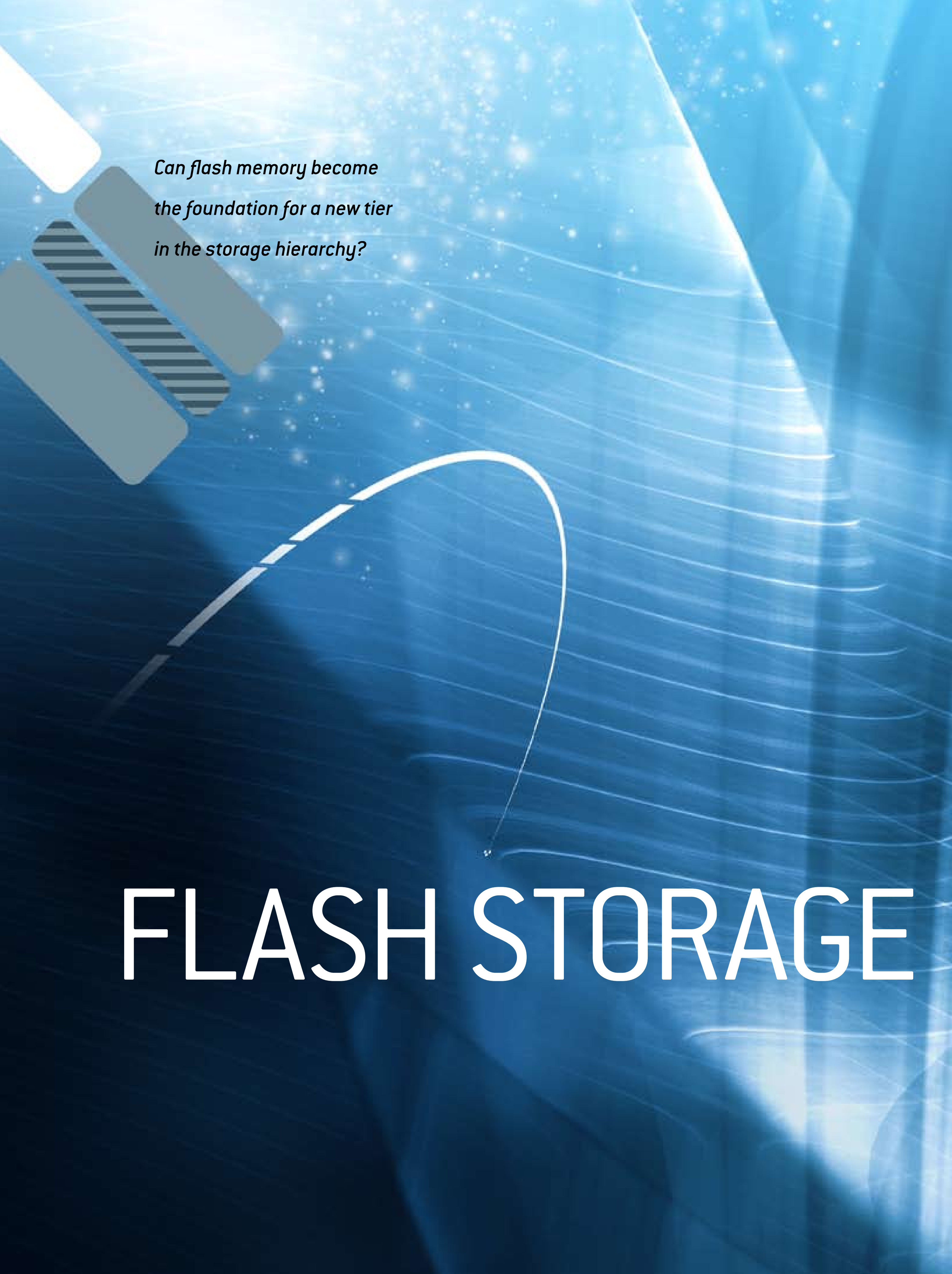
Can flash memory become the foundation for a new tier in the storage hierarchy?

FLASH STORAGE



The past few years have been an exciting time for flash memory. The cost has fallen dramatically as fabrication has become more efficient and the market has grown; the density has improved with the advent of better processes and additional bits per cell; and flash has been adopted in a wide array of applications. The flash ecosystem has expanded and continues to expand—especially for thumb drives, cameras, ruggedized laptops, and phones in the consumer space.

One area where flash has seen only limited success, however, is in the primary-storage market. As the price trend for flash became clear in recent years, the industry anticipated its ubiquity for primary storage, with some so bold as to predict the impending demise of rotating media (undeterred, apparently, by the obduracy of magnetic tape). Flash has not lived up to these high expectations, however. The brunt of the effort to bring flash to primary storage has taken the form of SSDs (solid-state disks), flash memory packaged in hard-drive form factors and designed to supplant conventional drives. This technique is alluring because it requires no changes to software or other hardware components, but the cost of flash per gigabyte, while falling quickly, is still far more than hard drives. Only a small number of applications have performance needs that justify the expense.

Although flash's prospects are tantalizing, the challenge is to find uses for it that strike the right balance between cost and performance. Flash should be viewed not as a replacement for existing storage, but rather as a means to enhance it. Conventional storage systems mix

Today

Adam Leventhal,
Sun Microsystems

FLASH STORAGE Today

DRAM (dynamic memory) and hard drives; flash is interesting because it falls in a sweet spot between those two components for both cost and performance in that flash is significantly cheaper and denser than DRAM and significantly faster than disk (see figure 1). Flash can accordingly augment the system to form a *new tier* in the storage hierarchy—perhaps the most significant new tier since the introduction of the disk drive with RAMAC in 1956.

PROPERTIES OF FLASH

Flash has two distinct categories: NAND and NOR—designations that refer to the way the flash cells are arranged. NOR flash allows for random access and is best suited for random access memory, while NAND must be treated as blocks and is ideal for persistent storage. The rest of this article examines only NAND flash, the cheaper and more common variety, of which again there are two types: SLC (single-level cell) and MLC (multilevel cell). SLC stores a single binary value in each memory cell. The binary value is distinguished by two threshold voltages. MLC supports four or, recently, eight distinct values per memory cell, corresponding to two or three bits of storage. Because of its improved longevity and performance, the conventional wisdom is that SLC is best suited for enterprise (i.e., nonconsumer-grade) solutions, so our focus here is on SLC flash, its cost, power dissipation, performance, and longevity as compared with DRAM and disk drives.

The cost per unit of storage is what has brought flash to the forefront in recent years (see figure 2). Earlier this decade, flash costs were on par with those of DRAM; now, flash devices are much less expensive: $10-$35 per gigabyte for an SLC flash device compared with around $100 per gigabyte for DRAM. The cost trend appears to be continuing to widen the gap between flash and DRAM. Disk drives are still much cheaper than flash, weighing in at less than $1 per gigabyte for 7200-RPM drives and in the neighborhood of $3 per gigabyte for 15,000-RPM drives.

The other exciting attribute of flash is its low power consumption. As the cost of power and the impetus toward green computing rise, so does the attractiveness of lower-power solutions. While completely accurate comparisons among flash, DRAM, and hard drives are difficult because of differences in capacity and interfaces, it's fair to say that flash consumes significantly less power than those other system components, especially on a per-gigabyte basis. Table 1 records the power consumption for some typical components to provide a broad sense for each type of device.

FIGURE 1 Price and Performance

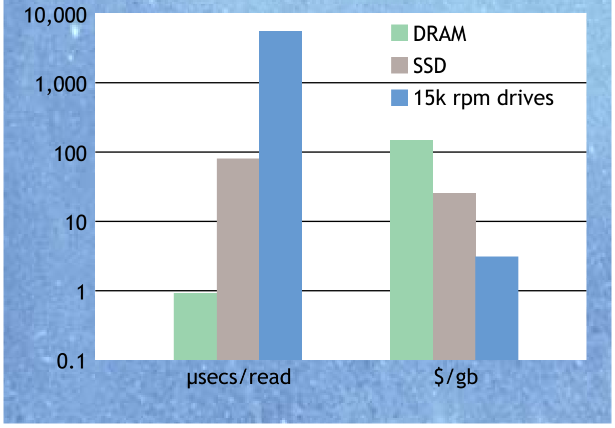
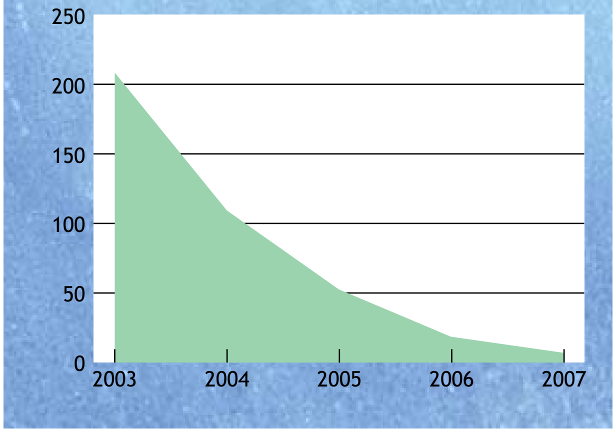


FIGURE 2 Recent Cost of Flash per GB



The performance of flash is a bit unusual in that it's highly asymmetric, posing a challenge for using it in a storage system. A block of flash must be erased before it can be written, which takes on the order of 1-2 ms for a block, and writing to erased flash requires around 200-300 µs. For this reason flash devices try to maintain a pool of previously erased blocks so that the latency of a write is just that of the program operation. Read operations are much faster: approximately 25 µs for 4k. By comparison, raw DRAM is even faster, able to perform reads and writes in much less than a microsecond. Disk-drive latency depends on the rotational speed of the drive: on average 4.2 ms for 7200 RPM, 3 ms for 10,000 RPM, and 2 ms for 15,000 RPM. Adding in the seek time bumps these latencies up an additional 3-10 ms depending on the quality of the mechanical components.

SLC flash is typically rated to sustain 1 million program/erase cycles per block. As flash cells are stressed, they lose their ability to record and retain values. Because of the limited lifetime, flash devices must take care to ensure that cells are stressed uniformly so that "hot" cells don't cause premature device failure. This is done through a process known as *wear leveling*. Just as disk drives keep a pool of spare blocks for bad-block remapping, flash devices typically present themselves to the operating system as significantly smaller than the amount of raw flash to maintain a reserve of spare blocks (and pre-erased blocks for performance). Most flash devices are also capable of estimating their own remaining lifetimes so systems can anticipate failure and take prophylactic action.

THE STORAGE HIERARCHY OF TODAY

Whether over a network or for local access, primary storage can be succinctly summarized as a head unit containing CPUs and DRAM attached to drives either in storage arrays or JBODs (just a bunch of disks). The disks are the primary repository for data—typical modern data sets range from a few hundred gigabytes up to a petabyte or more—while DRAM acts as a very fast cache. Clients communicate via read and write operations. Read operations are always synchronous in that the client is blocked until the operation is serviced, whereas write operations may be either synchronous or asynchronous depending on the application. For example, video streams may write data blocks asynchronously and verify only at the end of the stream that all data has been quiesced; databases, however, typically use synchronous writes to ensure that every transaction has been committed to stable storage.

On a typical system, the speed of a synchronous write is bounded by the latency of nonvolatile storage, as writes must be committed before they can be acknowledged. Read operations first check in the DRAM cache providing very low-latency service times, but cache misses must also wait for the slow procession of data around the spindle. Since it's quite common to have working sets larger than the meager DRAM available, even the best prefetching algorithms will leave many read operations blocked on the disk.

A brute-force solution for improving latency is simply to spin the platters faster to reduce rotational latency, using 15,000-RPM drives rather than 10,000- or 7,200-RPM drives. This will improve both read and write latency, but only by a factor of two or so. For example, a 10-TB data set on a 7,200-RPM drive (from a major vendor, at current prices) would cost about $3,000 and dissipate 112 watts; the same data set on a 15,000-RPM drive would cost $22,000 and dissipate 473 watts—all for a latency improvement of a bit more than a factor of two. The additional cost and power overhead make this an unsatisfying solution, though it is widely employed absent a clear alternative.

A focused solution for improving the performance of synchronous writes is to add NVRAM (nonvolatile RAM) in the form of battery-backed DRAM, usually on a PCI card. Writes are committed to the NVRAM ring buffer and immediately acknowledged to the client while the data is asynchronously written out to the drives. Once the data has been committed to disk, the corresponding record can be freed in the NVRAM. This technique allows for a tremendous improvement for synchronous writes, but suffers some downsides. NVRAM is quite expensive; batteries fail (or leak or, worse,

TABLE 1 **Power Consumption Comparison**

Device	Approximate power consumption
DRAM DIMM module (1 GB)	5W
15,000-RPM drive (300 GB)	17.2W
7200-RPM drive (750 GB)	12.6W
High-performance flash SSD (128 GB)	2W

explode); and the maximum size of NVRAM tends to be small (2-4 GB)—small enough that workloads can fill the entire ring buffer before it can be flushed to disk.

FLASH AS A LOG DEVICE

One use of flash is as a stand-in for NVRAM that can improve write performance as a log device. To that end you need a device that mimics the important properties of NVRAM (fast, persistent writes), while avoiding the downsides (cost, size, battery power). Recall, however, that while achieving good write *bandwidth* is fairly easy, the physics of flash dictate that individual writes exhibit relatively high *latency*. It's possible, however, to build a flash-based device that can service write operations very quickly. This is done by inserting a DRAM write cache and then treating it as nonvolatile by adding a supercapacitor that, in case of power loss, provides the necessary power to flush outstanding data in the DRAM to flash.

Many applications, such as databases, can use a dedicated log device as a way of improving the performance of write operations; for these applications, such a device can be easily dropped in. To bring the benefits of a flash log device to primary storage, and therefore to a wide array of applications, we need similar functionality in a general-purpose file system. Sun's ZFS provides a useful context for the use of flash. ZFS, an enterprise-class file system designed for the scale and requirements of modern systems, was implemented from scratch starting in 2001. It discards the model of a file system sitting on a volume manager in favor of pooled storage for both simplicity of management and greater flexibility for optimizing performance. ZFS maintains its on-disk data structures in a way that is always consistent, eliminating the need for consistency checking after an unexpected power failure. Furthermore, it is flexible enough to accommodate new technological advances, such as new uses of flash. (For a complete description of ZFS, see http://opensolaris.org/os/community/zfs.)

ZFS provides for the use of a separate intent-log device (a *slog* in ZFS jargon) to which synchronous writes can be quickly written and acknowledged to the client before the data is written to the storage pool. The slog is used only for small transactions, while large transactions use the main storage pool—it's tough to beat the raw throughput of large numbers of disks. The flash-based log device would be ideally suited for a ZFS slog. The write buffer on the flash device has to be only large enough to saturate the bandwidth to flash. Its DRAM size requirements—and therefore the power requirements—are quite small. Note also that the write buffer is much smaller than the required DRAM in a battery-backed NVRAM device. There are effectively no constraints on the amount of flash that could be placed on such a device, but experimentation has shown that 10 GB of delivered capacity is more than enough for the vast majority of use cases.

Using such a device with ZFS in a test system, we measured latencies in the range of 80-100 μs. This approaches the performance of NVRAM and has many other benefits. A common concern about flash is its longevity. SLC flash is often rated for 1 million write/erase cycles, but beyond several hundred thousand, the data-retention period can drop to just a few weeks. ZFS will write to this device as a slog in 8-KB chunks with each operation taking 80 μs. On a device with 10 GB of raw flash, this equates to about 3½ years of constant use. A flash device with a formatted capacity of 10 GB will, however, typically have 20-50 percent more flash held in reserve, easily taking the longevity of such a device under constant use to five years. The device itself can report its expected remaining lifetime as it counts down its dwindling reserve of spare blocks. Further, data need be retained only long enough for the system to recover from a fatal error; a reasonable standard is 72 hours, so a few weeks of data retention, even for very old flash cells, is more than adequate and a vast improvement on NVRAM.

FLASH AS A CACHE

The other half of this performance picture is read latency. Storage systems typically keep a DRAM cache of data the system determines a consumer is likely to access so that it can service read requests from that cache rather than wait for the disk. In ZFS, this subsystem is called the ARC (adaptive replacement cache). The policies that determine which data is present in the ARC attempt to anticipate future needs, but read requests can still miss the cache as a result of bad predictions or because the working set is simply larger than the cache can hold—or even larger than the maximum configurable amount of DRAM on a system. Flash is well suited for acting as a new second-level cache between memory and disk in terms of capacity and performance. In ZFS, this is called the L2ARC.

ZFS fills the L2ARC using large asynchronous writes and uses the cache to satisfy read requests seamlessly from clients. The requirements here are a perfect fit for flash, which inherently has sufficient write bandwidth and fantastic read latency. Since these devices can be external—rather than being attached to the main board, as is the case with DRAM—the size of the L2ARC is limited only by the amount of DRAM required for bookkeeping (at a ratio of 50:1 in the current ZFS implementation). For example, the maximum memory configuration on a four-socket machine is usually around 128 GB; such a system can easily accommodate 768 GB or more using flash SSDs in its internal drive bays. ZFS's built-in checksums catch cache inconsistencies and mean that defective flash blocks simply lead to fewer cache hits rather than data loss.

In the context of the memory hierarchy, caches are often populated as entries are evicted from the previous layer—in an exclusive cache architecture, on-chip caches are evicted to off-chip caches, etc. With a flash-based cache, however, the write latency is so poor that the system could easily be bogged down waiting for evictions. Accordingly, the L2ARC uses an *evict-ahead* policy: it aggregates ARC entries and predictively pushes them out to flash, thus amortizing the cost over large operations and ensuring that there's no additional latency when the time comes to evict an entry from the ARC. The L2ARC iterates over its space as a ring, starting back at the beginning once it reaches the end, thereby avoiding any potential for fragmentation. Although this technique does mean that entries in the L2ARC that may soon be accessed could be overwritten prematurely, bear in mind that the hottest data will still reside in the DRAM-based ARC. ZFS will write to the L2ARC slowly, meaning that it can take some time to warm up; but once warm, it should remain so, as long as the writes to the cache can keep up with data churn on the system.

It's worth noting that up to this point the L2ARC hasn't even taken advantage of what is usually considered to be a key feature of flash: nonvolatility. Under normal operation, the L2ARC treats flash as cheap and vast storage. As it writes blocks of data to populate the cache devices, however, the L2ARC includes a directory so that after a power loss, the contents of the cache can be identified, thus *pre-warming* the cache. Although resets are rare, system failures, power failures, and downtime because of maintenance are all inevitable; the instantly warmed cache reduces the slow performance ramp typical of a system after a reset. Since the L2ARC writes slowly to its flash devices and data on the system may be modified quite quickly (especially with the use of flash as a log device), the contents of the L2ARC may not reflect the same data as is stored on disk. During normal operation, dirtied and stale entries are marked as such so that they are ignored. After a system reset, though stale data may be read off the cache device, metadata kept on the device and ZFS's built-in checksums are used to identify this condition and seamlessly recover by reading the correct data from disk.

Most of the time, caching even a very large working set in flash is cheaper than storing all data on fast disks.

FLASH STORAGE Today

For a working set that is larger than the DRAM capacity, flash offers an avenue to access that working set much faster than could otherwise be done by disks of any speed. Even for a working set that could comfortably fit in DRAM, if the absolute performance of DRAM isn't necessary, it may be more economical to skimp on DRAM for the main ARC and instead cache the data on flash. As this use of flash meshes perfectly with its natural strengths, suitable devices can be produced quite cheaply and still have a significant performance advantage over fast disks. Although flash is still more expensive than fast disks per unit of storage, most of the time caching even a very large working set in flash is cheaper than storing all data on fast disks.

THE IMPACT OF FLASH

By combining the use of flash as an intent-log to reduce write latency with flash as a cache to reduce read latency, we can create a system that performs far better and consumes less power than other systems of similar cost. It's now possible to construct systems with a precise mix of write-optimized flash, flash for caching, DRAM, and cheap disks designed specifically to achieve the right balance of cost and performance for any given workload, with data automatically handled by the appropriate level of the hierarchy. It's also possible to address specific performance problems with directed rather than general solutions. Through the use of smarter software, we can build systems that integrate different technologies to extract the best qualities of each. Further, the use of smarter software will allow flash vendors to build solutions for specific problems rather than gussying up flash to fit the anachronistic constraints of a hard drive. ZFS is just one example among many of how one could apply flash as a log and a cache to deliver total system performance. Most generally, this new flash tier can be thought of as a radical form of HSM (hierarchical storage management) without the need for explicit management.

Although these solutions offer concrete methods of integrating flash into a storage system, they also raise a number of questions and force us to reconsider many aspects of the system. For example, how should we connect flash to the system? SSDs are clearly an easy approach, but there may be faster interfaces, such as the memory bus. More broadly, how will this impact the balance of a system? As more requests are serviced from flash, it may be possible to provision systems with far more network connectivity to clients than bus connectivity to disks.

In that vein, flash opens the possibility of using disks that are even slower, cheaper, and more power efficient. We can now scoff at a 15,000-RPM drive as an untargeted half-measure for a variety of problems, but couldn't the same argument be applied to a 7200-RPM drive? Just because it's at the low end of the performance curve doesn't mean it's at the bottom. The 5400-RPM drive is quite common today and consumes less power still. Can the return of the 3600-RPM drive be far behind? The cost of power has continued to rise, but even if that trend were to plateau, a large portion of the total cost of ownership of a storage system is directly tied to its power use—and that's to say nothing of the increased market emphasis on green design. Flash provides solutions that require us to rethink how we build systems and challenge us to develop smarter software to drive those systems; the result will be faster systems that are cheaper and greener. **Q**

ACKNOWLEDGMENTS

Props to Neil Perrin for developing slogs, to Brendan Gregg for developing the L2ARC, and to Jeff Bonwick and Matt Ahrens for reinventing storage with ZFS. I would also like to thank Bryan Cantrill, Matt Amdur, and Beverly Hodgson for their diligent review of this article.

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

ADAM LEVENTHAL is a staff engineer on Sun's Fishworks advanced product development team. He is one of the three authors of DTrace, for which he and his colleagues received Sun's Chairman's Award for Technical Excellence in 2004. He was named one of *InfoWorld*'s Innovators of 2005 and won top honors from the 2006 *Wall Street Journal*'s Innovation Awards. Leventhal joined Sun after graduating cum laude from Brown University in 2001 with a degree in math and computer science.

This article appeared in print in the July 2008 issue of Communications of the ACM *Vol. 51, No. 7*.