Position Summary: The *Conquest* File System—Life after Disks

An-I A. Wang, Peter Reiher, and Gerald J. Popek[✧]
Computer Science Department
University of California, Los Angeles
{awang, reiher, popek}@fmg.cs.ucla.edu

Geoffrey H. Kuenning
Computer Science Department
Harvey Mudd College
geoff@cs.hmc.edu

The cost of paper and film has been a critical barrier to cross for a storage technology to achieve wide deployment and better economy of scale. By 2003, the declining cost of persistent RAM (e.g., battery-backed DRAM) will break this barrier, signifying the arrival of the persistent-RAM-based storage era.

Persistent RAM will not fully replace disks for many years. However, as RAM becomes cheap, memory can assume more roles of file systems. In particular, by 2005 high-end desktops can afford to be equipped with 4 to 10 Gbytes of persistent RAM for storage; this is sufficient to deliver nearly all aspects of file-system services, with the single exception of high-capacity storage.

The *Conquest* file system is designed to provide a transition from disk- to persistent-RAM-based storage. Initially, we assume 2 to 4 Gbytes of persistent RAM and the popular single-user desktop environment. Unlike other memory file systems, *Conquest* can incrementally assume more responsibility for in-core storage as memory prices decline. The *Conquest* approach realizes most of the benefits of persistent-RAM-based file systems before persistent RAM becomes cheaply abundant. *Conquest* also benefits from the removal of disks as the primary storage by identifying disk-related complexities and isolating them from the critical path where possible.

Unlike cache, which treats main memory as a scarce resource, *Conquest* anticipates the abundance of cheap persistent RAM. *Conquest* uses disk to store only the data well suited for disk characteristics. Reducing the range of access patterns and characteristics anticipated by the file system translates into simpler disk optimizations.

Our initial *Conquest* implementation uses core memory to store all metadata, small files (currently based on a size threshold), executables, and dynamically linked libraries, leaving only the content of the large files on disk. All accesses to in-core data and metadata incur no data duplication or disk-related overhead, and executions are in-place. For the large-file-only disk storage, we can use a larger access granularity to reduce the seek-time overhead. Because most accesses to large files are sequential, we can relax many historical disk design constraints, such as complex layout heuristics intended to reduce fragmentation or average seek times.

Conquest also speeds up computing by allowing easy reuse of previously computed results. With an expanded API, *Conquest* allows direct storage of runtime data structures, states, or even processes that interact with the environment in constrained ways. Unlike memory-mapped files, storing runtime states under *Conquest* requires no compaction or alignment to page boundaries, which benefits many data representations. Direct storage of runtime states relieves developers of the need for serialization and deserialization. Applications can also take advantage of storing runtime data in the most appropriate form for processing. For example, network applications can store outbound data in the format of network packets to bypass both disk-to-memory and memory-to-network translations.

Storing data in core inevitably invites the question of reliability and data integrity. However, conventional techniques of sandboxing, access control, checkpointing, fsck, and object-oriented self-verification still apply. For example, *Conquest* still needs to perform frequent system backups. *Conquest* uses common memory protection mechanisms by having a dedicated memory address space for storage (assuming a 64-bit address space). A periodic fsck is still necessary, but it runs at memory speed. We are also exploring the object-store approach of having a "typed" memory area, so a pointer can be verified to be of a certain type before dereferencing.

Various areas of *Conquest* are under investigation. Memory under *Conquest* is a shared resource among execution, storage, and buffering for disk access. Finding the "sweet spot" for system performance requires both modeling and empirical investigation. The ability for *Conquest* to store runtime states has the flavor of wide-address-space computing, which can be applied and extended to the distributed environment and database systems.

The *Conquest* prototype is operational under Linux 2.4.2. It is POSIX compliant and supports both in-core and on-disk storage. The source consists of 3,800 lines of kernel code, 1,400 lines of file-system-creation code, and 3,600 lines of testing code. Initial deployment and performance measurements are under way.

[✧] Gerald Popek is also associated with NetZero, Inc.