# Exofiles: Unbundling the Filesystem

Martin Swany, Indiana University

## 1 Introduction

Filesystems are one of the key services provided by operating systems and the notion of a "file" has been used as an abstraction for storage or memory since the 1950s. Managing and organizing storage in this way is common and arguably natural. Programmer productivity is aided by thinking of the abstraction of "files" in a hierarchy of folders and perhaps filesystems or file cabinets.

The filesystem is perhaps the most important OS service with this abstraction even being used to expose other aspects of the OS mechanism and to provide *e.g.* interprocess communication. However, copying file data through the kernel is inefficient. We know that the messaging part of a high-performance application should minimize data copying, with the ideal being "zero-copy" operation. Clearly the management of storage infrastructure, devices and memory, is necessary but the traditional way in which this is realized can lead to inefficiencies. Although aspects like caching can improve performance, the OS must manage this caching with heuristics, which inevitably leads to suboptimal efficiency due at least in part to being decoupled from the application.

Our assertion is that the implementation and abstractions of files and filesystems must be fundamentally revisited for extreme-scale computing environments. In particular, we believe that the filesystem should be "unbundled" from the OS and moved to a runtime system, creating the possibilities for customization and scalability that we believe are essential moving forward.

## 2 What is a file?

A file is a set of objects, generally an ordered set of bytes, with higher-level elements being realized atop the byte array abstraction. A file has metadata about the group of bytes, with the most basic metadata being a name. Object stores or record-oriented files can be implemented as structure atop a byte array. The byte abstraction may be implemented atop a block or record abstraction by mapping a linear address to a block or record plus an offset and in most instances the underlying storage for a file consists of an array of blocks.

A file is also defined by "structural" metadata, which includes the location of the file's data in storage elements. In the common case of block-based storage, the structural metadata includes the location of the blocks that comprise the file. It is also important to consider what we can call "operational" metadata, which includes ephemeral structural metadata. This encompasses aspects like the current file pointer and the location of file blocks in the buffer cache as well as access accounting information.

These various aspects of metdata management are at the heart of the file and filesystem services provided by the OS. The metadata about the file data is potentially itself a significant amount of data and there are performance concerns throughout.

## 3 Unbundled Filesystems

The notion of "unbundling" the filesystem is not new. Moving OS functionality from a "monolithic" kernel to user-space subsystems is the core of the "microkernel" model. Even more dramatically different from traditional models is the "unbundling" of OS services into runtimes and libraries that can be linked into applications.

Revisiting filesystem models is particularly important in extreme scale systems. There is a growing recognition that filesystem functionality isn't one-model-fits-all. High performance applications may very well need to select functionality *a la carte* and only pay the overhead for the functionality that is required, when it is required. Filesystems can be customized based on the needs of the application, the type of the data, and the operating environment.

### 3.1 Exposing Parallelism

One of the most obvious ways in which unbundled filesystem impact extreme scale, data intensive systems is in exposing parallelism. Even in the case of parallel I/O libraries and parallel filesystems, the interface to the kernel exposes utilizes a serial abstraction.

### 3.2 Metadata Management

Metadata requirements change over the file's lifetime. There is a fundamental tradeoff in volume of metadata and storage management. The use of larger blocks reduces the size of metadata but introduces internal fragmentation and false sharing. Rather than make the selection for the entire filesystem, we can gain additional flexibility and efficiency by changing the metadata over time moving from extensible verbose metadata supporting long term storage, annotations and provenance to in-memory representations that are compact, fine-grained, hierarchical, and implicit.

## 4 Ongoing Efforts

We are approaching these ideas from various directions in two major efforts. In one sense, these represent efforts focused on extreme scale computing on the one hand and big data on the other. The convergence of these efforts enables further investigation of unbundling filesystem services, *i.e.* the Exofile model.

### 4.1 Photon I/O

Photon leverages remote direct memory access (RDMA) to move parallel I/O through intermediate nodes, selecting between *PUT* and *GET* models as demand and availability changes. Data can be coalesced progressively as it moves through the data pipeline. Photon enables construction of programs in terms of their dataflow requirements moving toward data-driven execution models that have the potential to be transformative.

### 4.2 Data Logistics Toolkit

The Data Logistics Toolkit is a storage infrastructure project (with M. Beck, T. Moore and P. Sheldon) based on the Internet Backplane Protocol (IBP) and an extensible structure metadata structure called the eXnode. IBP has been used to construct burst buffers to temporarily absorb and distribute large datasets as well as to create open content distribution networks for scientific data.