Revisiting Storage Programming Models

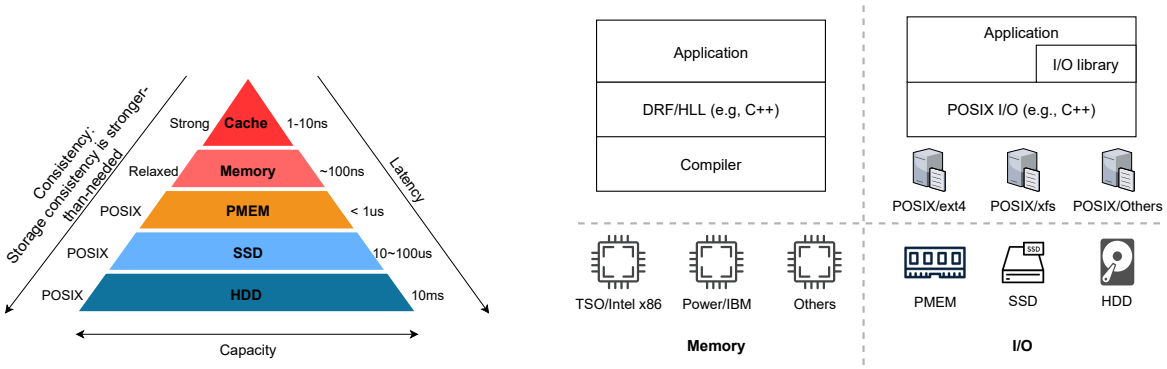
Chen Wang (chenw5@illinois.edu) and Marc Snir (snir@illinois.edu)
Department of Computer Science
University of Illinois at Urbana-Champaign

Topic

The use of the POSIX consistency model for I/O has plagued the HPC community for many years, but it is becoming more problematic due to two key reasons: (1) the rapid increase in the scale of HPC systems; (2) the emergence of the new storage techniques such as persistent memory. This problem can no longer be ignored especially as we move toward the exascale era. Even though POSIX consistency is the issue, the solution is not finding a better consistency model. The right solution is a paradigm shift away from the current consistency-centric I/O programming model to a synchronization-centric I/O programming model.

Challenge

There are two fundamental issues with the use of POSIX I/O for HPC: (1) It is overused; and (2) Its strict consistency model is a major bottleneck. The left edge of the pyramid below depicts the strictness of the consistency models provided at each storage level. One would expect that from top to bottom, the consistency model should be weaker and weaker as sharing becomes less frequent and access patterns become simpler. However, most storage systems only utilize the strong consistency model imposed by POSIX. Our previous study [4] has shown that HPC applications do not require the POSIX consistency model. A weaker consistency model can be used to improve performance without sacrificing programmability or portability.



(a) The most commonly used storage techniques

(b) Memory vs. I/O programming model

Figure 1: Memory vs. I/O

When it comes to the domain of memory, an important factor in the adoption of relaxed memory models is that compilers can hide the complexity. As shown in Figure 1(b), programmers target a single consistency model specified by the high-level programming language (e.g., C++ and Java) without the knowledge of underlying consistency models provided by the CPUs. In comparison, there is no corresponding "compiler" layer in the I/O programming model. General file systems (local or parallel) must provide the same consistency model (POSIX) regardless of the underlying storage hardware, which leads to unnecessarily reduced performance. Besides, the POSIX consistency is very expensive to maintain especially in distributed systems.

Opportunity

Several efforts [1, 2, 3, 5] have been made to alleviate the bottleneck caused by POSIX. These efforts propose to replace the POSIX PFSs with relaxed-semantics or tunable consistency PFSs as shown in Figure 2(a) and (b). Although these approaches have shown performance improvement, they are not long-term solutions to

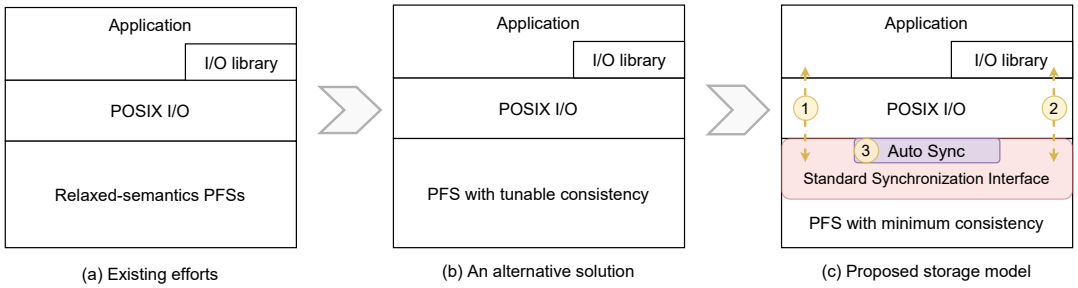


Figure 2: Both (a) and (b) are based on the current consistency-centric programming model, where PFSs are unaware of the application's synchronization logic and thus conservative consistency models have to be used. The proposed synchronization-centric programming model is shown in (c).

the fundamental issues of using the POSIX I/O model. The fact remains that there is no single consistency model that is optimal for all applications and hardware. Fundamentally, HPC I/O performance is hindered by the current consistency-centric programming model and we must change our approach.

We propose a shift away from the consistency-centric I/O model we use today in HPC to a synchronization-based I/O model, shown in Figure 2 (c). The core component is a standard synchronization interface. In contrast to the current programming model where file systems implement a standard consistency model (POSIX), here they implement a standard synchronization interface. File systems can provide whatever consistency models work the best for the underlying hardware, and HPC application programmers will utilize synchronization operations to ensure I/O correctness. We envision three ways to achieve the "proper synchronization" as denoted by the circled numbers in the figure:

1. Applications directly make use of the APIs provided by the standard synchronization interface. This should provide the best performance since application users know exactly where and when synchronization operations are needed. The drawback is that it requires modifications to the existing applications.
2. High-level I/O libraries or special synchronization libraries can provide an abstract layer to simplify the process of inserting synchronization operations. For example, they may allow annotations to existing function calls, e.g., marking a `close` function call as a synchronization operation. This method should be easier to use and can provide comparable performance to the first method.
3. Synchronization middleware can be designed to perform automatic synchronizations. The simplest implementation is to synchronize at every I/O operation. Both applications and high-level libraries can utilize this middleware. Since the application's I/O logic is unknown to the middleware, unnecessary synchronizations may be inserted which will lead to reduced performance. The advantage of this method is it requires little or no modification to the existing code.

We believe this new I/O programming model will have a significant impact on HPC systems and applications. We anticipate that many HPC applications will gain great performance improvement without any code changes. More importantly, the I/O bottleneck caused by the overuse of POSIX consistency can be addressed by this synchronization-centric programming model. With proper and accurate synchronizations, HPC applications should be ready to take advantage of future exascale storage systems.

References

[1] L. L. N. Laboratory. UnifyFS: A File System for Burst Buffers. `https://github.com/LLNL/UnifyFS`, Dec. 2021.

[2] O. Tatebe, S. Moriwake, and Y. Oyama. Gfarm/BB—Gfarm File System for Node-Local Burst Buffer. *Journal of Computer Science and Technology*, 35(1):61–71, 2020.

[3] M.-A. Vef, N. Moti, T. Süß, T. Tocci, R. Nou, A. Miranda, T. Cortes, and A. Brinkmann. GekkoFS: A Temporary Distributed File System for HPC Applications. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 319–324. IEEE, 2018.

[4] C. Wang, K. Mohror, and M. Snir. File System Semantics Requirements of HPC Applications. In *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, pages 19–30, 2020.

[5] T. Wang, K. Mohror, A. Moody, W. Yu, and K. Sato. BurstFS: A Distributed Burst Buffer File System for Scientific Applications. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2015.