# Unified High-Performance I/O: One Stack to Rule Them All

Animesh Trivedi[1], Patrick Stuedi[1], Bernard Metzler[1], Roman Pletka[1], Blake G. Fitch[2] and Thomas R. Gross[3]

[1]IBM Research, Zurich
[2]IBM Research, Yorktown Heights, NY
[3]ETH, Zurich

## Abstract

Fast non-volatile memories are exposing inefficiencies in traditional I/O stacks. Though there have been fragmented efforts to deal with the issues, there is a pressing need for a high-performance storage stack. Interestingly, 20 years ago, networks were faced with similar challenges, which led to the development of concepts and implementations of multiple high-performance network stacks. In this paper we draw parallels to illustrate synergies between high-performance storage requirements and concepts from the networking space. We identify common high-performance I/O properties and recent efforts in storage to achieve those properties. Instead of reinventing the performance wheel, we advocate a case for using mature high-performance networking abstractions and frameworks to meet the storage demands, and discuss opportunities and challenges that arise with this unification.

## 1 Introduction

Recent advancements in non-volatile memories (NVMs) promise to offer significant performance and endurance improvements over the current generation [2, 11]. However, despite having significant bandwidth and latency improvements in storage devices, the performance of end applications has only been improving at a marginal rate. One of the key reasons behind this inefficiency is retrofitting old storage interfaces and abstractions on top of modern NVM devices. This approach results in a centralized, CPU-centric I/O stack, where the CPU orchestrates data movements within the host. In a high-performance environment, the constant CPU involvement in data flow incurs high performance penalties [5, 7]. Stagnant single CPU speed improvements, multilayer device accesses in virtualized environments, absence of high-performance storage stacks, and inefficient APIs only exacerbate the situation.

To improve the situation, storage researchers have already started to look into light-weight, low-latency, asynchronous, and directly-accessible storage stacks [5, 6, 7]. Interestingly, looking back 20 years, similar challenges were faced by the networking community when traditional stacks were unable to meet strict application performance demands. The vast body of research into that resulted in defining concepts, interfaces, and concrete implementations of multiple high-performance networking stacks [3, 4, 10, 21].

Recent rejuvenated interest in efficient I/O stacks gives us opportunities to evaluate high-performance network abstractions and interfaces for storage. We present a case for a unified I/O stack, using high-performance networking framework. We discuss in detail the key characteristic properties, opportunities, required support and the open issues when applying networking concepts in the storage domain.

## 2 The Storage Performance Landscape

Slow storage has been the Achilles' heel for data processing systems. Historically disk bandwidth and access latency have consistently lagged behind its capacity and packing improvements [11]. However, Non-Volatile Memory offers unprecedented improvements over disks with multi-Gigabit bandwidth and access latencies in microseconds. This performance paradigm shift has been the most fundamental and significant change in storage since the advent of magnetic disks in the 70s. Unsurprisingly, not much has changed in the way operating systems manage storage devices. The following factors motivate a need for reevaluation of the complete storage stack in order to support high data rates:

**Rising CPU Gap:** Hardware landscape has changed considerably during the last decade. Modern I/O devices are getting significantly faster than CPUs. With stalled single CPU speed scaling, CPUs can no longer keep up with the high data rates from devices (see Table 1). As a

| | CPU Speed | Net BW | Storage BW |
|---|---|---|---|
| 1980-2010 | 1000× | 3000× | 50× |
| 2010-now | 1-1.5× | 4-10× | 10-100× |

Table 1: The widening CPU performance gap between a CPU speed and bandwidth improvements of I/O devices.

result, traditional CPU-centric storage stacks, where the CPU orchestrates data movement from relatively slow disks to DRAM buffers, have started to show performance strains in high IO operation/sec (IOPS) environments. This CPU-bottleneck limits deliverable performance to applications, despite having orders of magnitude performance improvements in hardware. As NVM technologies continue to mature, this performance gap between CPU and devices will widen. Manycore CPUs come to the rescue, however, the overhead due to locking, synchronization, and coherency etc., limits the overall achievable I/O performance. Also, to use multiple cores to satisfy high-CPU demands of I/O operations, is performance inefficient. As computing gradually moves toward Exascale, the performance efficiency [1] has direct implications for the amount of resources (CPU, storage, network), energy, and cost.

**Software and Access Overhead:** NVMs packed as *fast disks*, have been the least intrusive and most economical way of integration so far. As the performance characteristics of underlying storage media have changed significantly, the traditional disk-based optimizations are now considered expensive, obsolete, and intrusive. For example, I/O prefetching, buffer caching, request reordering and merging etc., all require additional time and CPU cycles (which are limited) and hence, may even lead to performance degradation with NVMs. Layering and multiplexing within operating systems, and virtualized environments put further penalties on the performance. With additional layers, the incurred overhead is non-linear which results in a rapid performance loss.

**Restrictive APIs:** Storage APIs are not designed to expose NVM capabilities to applications. Features such as accessing flash chips directly [14], parallel read/write ports [6], atomic updates [19] etc. can significantly simplify storage logic while improving deliverable I/O performance [16]. Furthermore, they also restrict passing useful information about the nature of I/O across the layers to NVM devices. Useful access information like scratch-pad access (light-weight, single copy, no protection), log-access (write-append, random reads), or range invalidation can help significantly with better device management, and consequently performance.

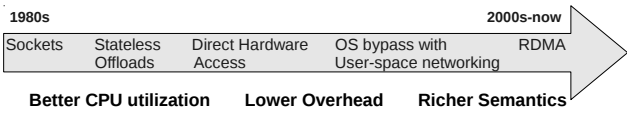


Figure 1: Evolution of high-performance network properties. The arrow does not represent any casual dependency or temporal precedence in the development.

## 3 A View from Networks

This appetite for efficient high-bandwidth and low-latency access to data is not unique to the storage. The networking community has always lived with stringent application demands for high IOPS. Over the last 20 years various techniques and concrete implementations of networking stacks have developed to match the periodic interconnect bandwidth and latency improvements.

Simple optimizations such as checksum and segmentation offloads gradually delegated a part of the packet generation to network controllers. Adaptive interrupt coalescing and device polling resulted in better device management under load. These simple techniques freed precious CPU cycles and helped to close the interim CPU-network gap that arose from continuous interconnect improvements e.g. Megabit to Gigabit Ethernet.

Though effective and helpful for high-bandwidth data transfers, these optimizations yielded little improvements in end-to-end application latencies. Latency requirements of high-performance applications necessitated more radical approaches. To reduce every potential overhead, these approaches favored a fresh redesign of the complete networking stack and developed novel host interfaces, interconnects, networking principles and operating system mechanisms [3, 4, 10, 21].

As these high-performance stacks became popular, network architects soon identified a common requirement for rich network I/O semantics from many applications. These semantics and network operations made development of complex applications easier. Naturally, to reflect the gradual progress made in operating systems and networking hardware, networking API and interfaces also evolved.

The holistic approach helped in developing many key ideas that are now an integral part of any modern high-performance interconnect such as Infiniband and iWARP. Given the recent rejuvenated interest in high performance I/O, this is a timely investigation into key principles, and properties that enabled efficient I/O for networks. Though these properties are inspired from experiences in networks, we argue that these are equally applicable to the storage domain as well. We discuss how recent efforts to integrate NVM are already exploring subsets of these properties (see Table 2):

| | Networks | Moneta-D | Gordon | NVHeap | FusionIO |
|---|---|---|---|---|---|
| Efficient Hardware Access | yes | yes | yes | yes | yes |
| Operating System Bypass | yes | yes | N/A | N/A | proprietary |
| Zero Copy Data Movement | yes | no | N/A | possibly | no |
| Asynchronous I/O Model | yes | yes | yes | N/A | yes |
| Synchronous Completion | yes | no | no | N/A | no |
| Rich I/O and API | yes | no | no | transactions | proprietary |

Table 2: Comparison of recent storage efforts to achieve high-performance properties. N/A denotes that the property is not the primary focus of the work.

**Efficient Host Interface and Hardware Access:** High-performance networks manage to keep host overhead minimal by *directly mapping the hardware resources* to applications as private channels or queues. As the overhead from disk based storage protocols (SCSI etc.) and host interfaces (AHCI etc.) become unbearable [9], research projects such as Moneta [5] and multiple commercial offerings [14] have started to look into directly accessible hardware with improved host interfaces. Moneta-D offers safe user-space access to directly accessible NVM devices [7]. Though it helps in reducing the overhead associated with issuing I/O requests and notification delivery, these efforts lack the generality of user-space networking.

**Operating System Bypass:** By separating data from the control path and pre-allocating I/O resources, high-performance networks involve operating systems in selective managerial tasks such as resource accounting. Enforcement of the security policies takes place in network hardware. Recent storage research has looked into similar techniques to avoid unnecessary operating system involvement with request scheduling, batching, reordering, dynamic resource allocation, and security enforcement. Moneta-D pre-allocates DMA buffers, directly posts requests, and offloads file permission checks to a capable storage hardware [7]. Though the operating system is still involved in DMA buffer management, file check offloading and permission evictions etc., it is kept out of the I/O loop between the application and hardware.

**Zero Copy Data Movement:** High-performance network controllers maintain sufficient contextual meta-data to multiplex, and securely DMA data directly into application buffers. They also support arbitrary application buffer layouts, offset calculations, and scatter-gather I/O. Together with directly accessible I/O hardware and operating system bypassing, the CPU is now completely decoupled from the fast data flow. Efforts have been made to achieve zero-copy storage, but they are either limited (small number of user accessible DMA buffers) or restricted (aligned layout of user buffers). Zero copy storage is possible with mmap'ed files as application buffers, but not achievable using other memory allocation methods such as malloc.

**Asynchronous I/O Posting:** High-performance storage interfaces such as epoll, are based upon the *readiness* instead of the *asynchronous-notification* model. This does not provide sufficient concurrency to exploit full device potential, and makes optimizations such as request batching and selective notifications, very difficult. In multi-stage environments, where data passes through multiple storage devices, asynchronous I/O also gives better I/O scheduling opportunities for a smooth end-to-end data flow. Recent efforts such as Moneta-D [7] and MegaPipe [15], have advocated the use of the asynchronous model with private channels.

**Synchronous Completion:** A synchronous completion allows posting I/O requests and reaping completion notifications without context switches. Networks support non-blocking posting of batch requests with polling completion within the same user-context. For low-latency networks, this method delivers better application latencies at the expense of higher CPU utilization. However, as emerging NVM device latencies will fall below context switch latencies, this approach proves to be more favorable for storage as well [22]. High-performance networks also allow applications to adaptively switch between blocking and polling for completion. It is a desired property for storage, but no attempts have been made to provide such functionality.

**Rich I/O Operations and APIs:** Modern interconnects support operations such as remote data read and writes, fencing, atomic compare and swap, atomic add, scatter-gather I/O etc. Such hardware primitives make complex application development simpler. Similar experiences are also reported by the storage researchers in [16, 19]. NVHeap [8] saves the heap-state of an active application on NVM in a novel way and provides transactional support to access it. As the NVM integration has been transparent, these approaches do not provide much control over I/O.

# 4 A Case for Unified High Performance Stack

The current unified abstraction of *files* is not aimed at high performance and has plenty of performance overhead due to the need for global synchronization, inefficient I/O memory management, and lack of useful hints in multi-core environments [15]. One potential solution is to redesign the complete storage stack from scratch. However, as modern high-performance networking stacks offer very mature and stable implementations of desired key high-performance properties (see Table 2), in this paper we propose using them to access and transfer data to/from NVM devices.

Similar to the networks, high-performance storage interfaces can provide directly user-space mapped hardware I/O queues or channels for request postings and completion notifications. Operating system and user-space device libraries provide support for setting up direct NVM device access from the user-space. The devices implement multiple I/O request opcodes and associated completion semantics.

The access abstraction is a simple byte-addressable storage address space. Byte-addressable abstraction does not impose any data structure and is expressive enough to support a wide variety of higher-level storage systems such as hierarchical file-systems, databases or object stores. These systems are responsible for the translation of higher-level storage objects, such as a file or a database column, to a specific device address range. After the translation, the data transfer happens directly between the device and user-space buffers. The data transfer is done in a similar manner to a remote memory read or write operation. The following factors further support our case for a unified stack for common device management and data access:

**Unified operating system support:** The unification blurs the traditional boundary between network and storage and enables common evolution of high-performance I/O frameworks. A single stack provides uniform I/O semantics and guarantees across multiple kinds of devices (see Figure 2). This unification also simplifies the implementation of I/O mechanisms inside an operating system. Both, network and storage, need capabilities to directly access hardware, use adaptive notifications (callbacks, blocking, polling etc.), share I/O memory management with applications etc. Looking beyond the performance properties, networking stacks also have everything from device detection, configuration management, capabilities discovery, efficient memory management etc. These services largely simplify the device management.

**Ready to use:** Any modern high-performance network stack implementation can be used as a drop-in replacement to access storage. The replacement allows storage to reuse interfaces, data structures, APIs, and even (up to a certain extent) concrete implementations of a stack! Many networking semantics have an immediate appeal for storage applications. Features such as remote read/writes can be used to access data from storage. Fencing and ordering among I/O operations ensures proper consistency guarantees (similar to a remote memory) for storage. Multiple storage devices can form a multi/broadcasting group to implement replication. QoS can be implemented by using multiple network traffic classes. Furthermore, end-to-end semantics of the interface/API ensures light-weight data access even in multi-layer access environments e.g. virtualization.

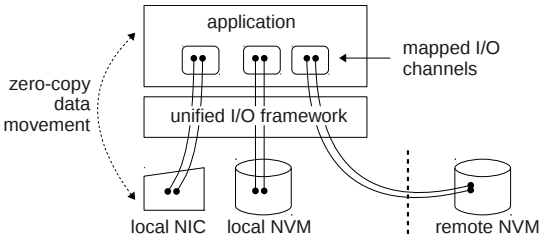


Figure 2: Illustration of the unified I/O stack that can deal with network, local NVMs, and remote NVMs under a common framework.

**Favorable advancements:** Lastly, recent architecture and systems advancements also facilitate this unification. The byte-addressability nature of NVMs (e.g. PCM) makes data access as simple as reading remote memory. This fits nicely with remote memory access semantics of RDMA. However, an NVM device itself does not have to natively support byte addressability as long as it understands the RDMA access model. With the revised host-interfaces [18], NVMs can now be directly accessed via networks, further blurring the gap between local and remote storage. Repartitioning storage responsibilities between application and hardware, also makes it possible to reuse standard user-space networking stacks.

# 5 Discussion

## 5.1 Operating System Support

As the key responsibilities of an operating system - abstraction, multiplexing, and layering - seem too prohibitive for high-performance, we must revise the responsibilities between devices and the operating system. Much of I/O management complexity from within operating systems can now be delegated to applications and hardware. Instead of micro-managing, an Exokernel [12] like the approach for OS design is more desirable. For example, instead of performing fine-grained I/O scheduling within an operating system, as has been done tra-

ditionally, it should be involved selectively in coarse-grained decisions such as when to schedule an I/O request class (e.g real-time, or backup) or controlling parallelism and concurrency within hardware for QoS. Hardware does a better job in fine-grained scheduling of individual I/O requests. Additionally, operating system developers must remove the stigma associated with I/O offloading. An operating system must be able to efficiently manage, extract, and communicate necessary contextual information about I/O to the devices.

Protection in a shared environment can be provided by generating access capabilities with the help of I/O devices. For example, an RDMA device generates an access identifier tag during the memory registration for a data buffer. This tag must be presented by an application for any access to the data buffer to verify access rights.

## 5.2 Hardware Support

Network and storage devices must be able to allocate I/O channels, generate protection identifiers, install user contexts and memory mappings etc. In order to be efficiently managed by the operating system, hardware vendors need to come up with a standard communication interface for device management and configuration such as openflow [17] for switches. Efforts have been made recently to standardize storage host-device interface [18, 20]. Interestingly, NVMExpress [18] shares many key performance properties with high performance networking devices. For example, directly accessible hardware resources, doorbell write to issue a command, multiple request, response queues, capability discovery, interrupt coalescing, configurable data block size etc.

To avoid unnecessary operating system involvement, devices must be educated about logical I/O primitives with gradually increasing complexity. As we discussed earlier, high-performance interconnects such as Infiniband, already support rudimentary forms of these operations such as atomic fetch and add, atomic compare and swap etc. We believe that with a minimal set of basic integrated operations (e.g. locking, logging, atomicity, serialization etc.), it should be possible to build higher-level complex primitives such as transactions or replication (using multicasting) without bloating the I/O stack.

## 5.3 Open Issues

**Multi-stage resource allocation:** Resource allocation in the control path of high-performance networks is a multi-stage process. Different I/O resources (with associated state) are allocated at various stages of a connection setup e.g. open channel, route discovery, device resolution, connect, accept etc. However, storage has a simple single-stage (e.g. open a file) access process. Reserving storage resources in a single step may lead to wasteful resource usage. In order to avoid overcommitment, additional access pattern and range related information must be passed to the storage. However, due to the lack of support in network interfaces to pass this information, it requires further development of new APIs.

**Hardware multiplexing:** Modern high-performance controllers can typically maintain 64K to 1M active contexts. However, high-level storage primitives such as files, can be in the billions. This will require some coarse-grained multiplexing support from the operating system such as the one found for virtual memory subsystem. By installing hardware contexts in the *storage page-table*, the operating system can move out of the way of normal I/O processing. This mechanism maintains the operating system control over I/O resources without sacrificing the performance. However, this functionality will require support from the storage hardware e.g. generating storage faults for an invalid access to files.

**File semantics:** Files are shared more often than sockets. Depending upon the mode, file sharing can lead to different consistency semantics. For example, accessing a shared file using a common request queue among multiple applications can potentially provide serialization guarantees but this may not be possible with different request queues or may require different I/O opcode. The packet oriented nature of network APIs makes development of stream-based storage applications difficult.

**I/O failures:** Direct-access zero-copy I/O has visible side-effects in the case of a failed operation. The byte-addressable nature of NVMs makes data corruption detection even harder. Hence, a more sophisticated and precise error reporting and cancellation framework is required. One possible solution is to maintain error and log data structures in DRAM, hence if there is a failure the operating system can still perform error diagnosis on it.

# 6 Conclusion

Storage stacks are at a familiar crossroad. Performance of I/O devices are improving at a much faster rate than a single CPU capacity. Over the last 20 years, networks have undergone an evolutionary transformation to support high-performance I/O. In this paper we argue that storage does not have to repeat the same steps as networks and wait a further 20 years to undergo the same transformation. Storage developers can directly use abstractions, frameworks, and interfaces developed by high-performance networks. This unification instantly enables efficient, light-weight high-IOPS access to NVM devices. The Blue Gene Active Storage project [13] explores the integration of Non-Volatile Memory and RDMA networks and could benefit from such an approach.

## References


[1] ANDERSON, E., AND TUCEK, J. Efficiency matters! *SIGOPS Oper. Syst. Rev. 44*, 1 (Mar. 2010), 40–45.

[2] BAILEY, K., CEZE, L., GRIBBLE, S. D., AND LEVY, H. M. Operating system implications of fast, cheap, non-volatile memory. In *Proceedings of the 13th USENIX HotOS* (2011).

[3] BLUMRICH, M. A., LI, K., ALPERT, R., DUBNICKI, C., FELTEN, E. W., AND SANDBERG, J. Virtual memory mapped network interface for the SHRIMP multicomputer. In *Proceedings of the 21st annual ISCA* (1994), pp. 142–153.

[4] BUZZARD, G., JACOBSON, D., MACKEY, M., MAROVICH, S., AND WILKES, J. An implementation of the Hamlyn sender-managed interface architecture. In *Proceedings of the second USENIX OSDI* (1996), pp. 245–259.

[5] CAULFIELD, A. M., DE, A., COBURN, J., MOLLOW, T. I., GUPTA, R. K., AND SWANSON, S. Moneta: A High-Performance Storage Array Architecture for Next-Generation, Non-volatile Memories. In *Proceedings of the 43rd MICRO* (2010), pp. 385–395.

[6] CAULFIELD, A. M., GRUPP, L. M., AND SWANSON, S. Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications. In *Proceedings of the 14th ASPLOS* (2009), pp. 217–228.

[7] CAULFIELD, A. M., MOLLOV, T. I., EISNER, L. A., DE, A., COBURN, J., AND SWANSON, S. Providing safe, user space access to fast, solid state disks. In *Proceedings of the 17th ASPLOS* (2012), pp. 387–400.

[8] COBURN, J., CAULFIELD, A. M., AKEL, A., GRUPP, L. M., GUPTA, R. K., JHALA, R., AND SWANSON, S. NV-Heaps: making persistent objects fast and safe with next-generation, non-volatile memories. In *Proceedings of the 16th ASPLOS* (2011), pp. 105–118.

[9] DON WALKER. A Comparison of NVMe and AHCI, white paper at `http://www.sata-io.org/documents/NVMandAHCI_(long).pdf`.

[10] DRUSCHEL, P., PETERSON, L. L., AND DAVIE, B. S. Experiences with a High-Speed Network Adaptor: A Software Perspective. In *SIGCOMM* (1994), pp. 2–13.

[11] ELEFTHERIOU, E., HAAS, R., JELITTO, J., LANTZ, M., AND POZIDIS, H. Trends in Storage Technologies. *IEEE Data Engineering Bulletin, 33(4), 4-13, IEEE* (2010).

[12] ENGLER, D. R., KAASHOEK, M. F., AND O'TOOLE, JR., J. Exokernel: an operating system architecture for application-level resource management. In *Proceedings of the 15th SOSP* (1995), pp. 251–266.

[13] FITCH, BLAKE G. AND OTHERS. Blue Gene Active Storage, at `http://institute.lanl.gov/hec-fsio/workshops/2010/presentations/day1/Fitch-HECFSIO-2010-BlueGeneActiveStorage.pdf`.

[14] FUSIONIO. Io drive specification, at `http://www.fusionio.com/products/iodrive/`.

[15] HAN, S., MARSHALL, S., CHUN, B.-G., AND RATNASAMY, S. MegaPipe: a new programming interface for scalable network I/O. In *Proceedings of the 10th USENIX OSDI* (2012), pp. 135–148.

[16] JOSEPHSON, W. K., BONGO, L. A., FLYNN, D., AND LI, K. DFS: a file system for virtualized flash storage. In *Proceedings of the 8th USENIX FAST* (2010).

[17] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S., AND TURNER, J. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.* (2008), 69–74.

[18] NVMEXPRESS. The Optimized PCI Express SSD Interface, at `http://www.nvmexpress.org/`.

[19] OUYANG, X., NELLANS, D., WIPFEL, R., FLYNN, D., AND PANDA, D. K. Beyond block I/O: Rethinking traditional storage primitives. In *Proc. of the 17th HPCA* (2011).

[20] SCSI EXPRESS. `http://www.scsita.org/library/scsi-express/`.

[21] VON EICKEN, T., BASU, A., BUCH, V., AND VOGELS, W. U-Net: a user-level network interface for parallel and distributed computing. In *Proceedings of the 15th SOSP* (1995), pp. 40–53.

[22] YANG, J., MINTURN, D. B., AND HADY, F. When poll is better than interrupt. In *Proceedings of the 10th USENIX FAST* (2012).