Decibel: Isolation and Sharing in Disaggregated Rack-Scale Storage

Mihir Nanavati Jake Wires Andrew Warfield
Coho Data and University of British Columbia

Abstract

The performance characteristics of modern non-volatile storage devices have led to storage becoming a shared, rack-scale resource. System designers have an imperative to rethink existing storage abstractions in light of this development. This paper proposes a lightweight storage abstraction that encapsulates full-system hardware resources and focuses only on isolating and sharing storage devices across multiple, remote tenants. In support of this, it prototypes a shared-nothing runtime capable of managing hardware and scheduling this abstraction in a manner that avoids performance interference and preserves the performance of the storage devices for tenants.

1 Introduction

Rack-scale storage architectures such as Facebook's Lightning [53] and EMC's DSSD [17] are dense enclosures containing storage class memories (SCMs)[1] that occupy only a few units of rack space and are capable of serving millions of requests per second across petabytes of persistent data. These architectures introduce a tension between efficiency and performance: the bursty access patterns of applications necessitate that storage devices be shared across multiple tenants in order to achieve efficient utilization [39], but the microsecond-granularity access latencies of SCMs renders them highly sensitive to software overheads along the datapath [12, 64].

This tension has forced a reconsideration of storage abstractions and raised questions about where functionality, such as virtualization, isolation, and redundancy, should be provided. How should these abstractions evolve to support datacenter tenants without compromising efficiency, performance, or overall system complexity?

Decibel is a thin virtualization platform, analogous to a processor hypervisor, designed for rack-scale storage, that demonstrates the ability to provide tenants with flexible, low-latency access to SCMs. Its design is motivated by the following three observations:

1. Device-side request processing simplifies storage implementations. By centralizing the control logic necessary for multi-tenancy at processors adjacent to storage devices, traditional storage systems impose latency overheads of several hundreds of microseconds to few milliseconds under load [39]. In an attempt to preserve device performance, there has been a strong trend towards bypassing the CPU altogether and using hardware-level device passthrough and proprietary interconnects to present SCMs as *serverless storage* [12, 27, 3].

Serverless storage systems throw the proverbial baby out with the bathwater – eliminating the device-side CPU from the datapath also eliminates an important mediation point for client accesses and shifts the burden of providing datapath functionality to client-based implementations [4, 11] or to the devices themselves [71, 27, 1]. For isolation, in particular, client implementations result in complicated distributed logic for co-ordinating accesses [27] and thorny questions about trust.

2. Recent datacenter infrastructure applications have encompassed functionality present in existing feature-rich storage abstractions. In addition to virtualizing storage, storage volumes provide a rich set of functionality such as data striping, replication, and failure resilience [51, 18, 9, 28]. Today, scalable, cloud-focused *data stores* that provide persistent interfaces, such as key-value stores, databases, and pub/sub systems, increasingly provide this functionality as part of the application; consequently, the provision of these features within the storage system represents a duplication of function and risks introducing both waste and unnecessary overheads.

3. Virtualizing only the capacity of devices is insufficient for multi-tenancy. Extracting performance from SCMs is extremely compute-intensive [72, 12, 50] and sensitive to cross-core contention [6]. As a result, storage systems require a system-wide approach to virtualization and must ensure both adequate availability of compute, network, and storage resources for tenant requests, and the ability to service these requests in a contention-free manner. Further, unlike traditional storage volumes that do not adequately insulate tenants from performance interference [39], the system must provide tenants with predictable performance in the face of multi-tenancy.

These observations guide us to a minimal storage abstraction that targets isolation and efficient resource sharing for disaggregated storage hardware. Decibel introduces *Decibel volumes* (referred to as *dVols* for short): vertical slices of the storage host that bind SCMs with the compute and network resources necessary to service

[1] We use the term storage class memory through the rest of the paper to characterize high performance PCIe-based NVMe SSDs and NVDIMM-based persistent storage devices.

tenant requests. As both the presentation of fine-grained storage abstractions, such as files, objects, and key-value pairs, and datapath functionality, such as redundancy and fault-tolerance, have moved up the stack, dVols provide a minimal consumable abstraction for shared storage without sacrificing operational facilities such as transparent data migration.

To ensure microsecond-level access latencies, Decibel prototypes a runtime that actively manages hardware resources and controls request scheduling. The runtime partitions hardware resources across cores and treats dVols as schedulable entities, similar to threads, to be executed where adequate resources are available to service requests. Even on a single core, kernel scheduling policies may cause interference, so Decibel completely bypasses the kernel for both network and storage requests, and co-operatively schedules request processing logic and device I/O on a single thread.

Decibel is evaluated using a commodity Xeon server with four directly-connected enterprise PCIe NVMe drives in a single 1U chassis. Decibel presents storage to remote tenants over Ethernet-based networking using a pair of 40GbE NICs and achieves device-saturated throughputs with a latency of 220-450µs, an overhead of approximately 20-30µs relative to local access times.

2 Decibel and dVols

Scalable data stores designed specifically for the cloud are important infrastructure applications within the datacenter. Table 1 lists some popular data stores, each of which treats VM or container-based nodes as atomic failure domains and handles lower-level network, storage, and application failures uniformly at the node level. As a result, several of these data stores recommend deploying on "ephemeral", locally-attached disks in lieu of reliable, replicated storage volumes [47, 40, 14].

These systems are designed to use simple local disks because duplicating functionality such as data redundancy at the application and storage layers is wasteful in terms of both cost and performance; for example, running a data store with three-way replication on top of three-way replicated storage results in a 9x write amplification for every client write. Further, running a replication protocol at multiple layers bounds the latency of write requests to the latency of the slowest device in the replica set. The desire to minimize this latency has led to the development of persistent key-value stores, such as LevelDB and RocksDB, that provide a simple, block-like storage abstraction and focus entirely on providing high performance access to SCMs.

The dVol is an abstraction designed specifically for rack-scale storage in response to these observations. As a multi-tenant system, Decibel faces challenges similar to a virtual machine monitor in isolating and sharing an extremely fast device across multiple tenants and benefits from a similar lightweight, performance-focused approach to multiplexing hardware. Correspondingly, a dVol is a schedulable software abstraction that encapsulates the multiple hardware resources required to *serve* stored data. In taking an end-to-end view of resource sharing and isolation rather than focusing only on virtualizing storage capacity, dVols resemble virtual machines to a greater degree than traditional storage volumes.

Application	Backend	FT	Ephem	Year
Key-Value Stores				
Riak	LevelDB	Y	Y	2009
Voldemort	BerkeleyDB	Y	N	2009
Hyperdex	File	Y	Y	2011
Databases				
Cassandra	File	Y	Y	2008
MongoDB	RocksDB	Y	N	2009
CockroachDB	RocksDB	Y	Y	2014
Pub/Sub Systems				
Kafka	File	Y	Y	2011
Pulsar	BookKeeper	Y	Y	2016

Table 1: **Examples of cloud data stores.** *FT* denotes that replication and fault-tolerance are handled within the data store and storage failures are treated as node failures. *Ephem* indicates that users are encouraged to install data stores over local, "ephemeral" disks – Voldemort and Mongo suggest using host-level RAID as a convenience for recovering from drive failures. Backend is the underlying storage interface; all of these systems assume a local filesystem such as ext4, but several use a library-based storage abstraction over the file system API.

In borrowing from VMs as a successful abstraction for datacenter computation, dVols provide the following properties for storage resources:

Extensible Hardware-like Interfaces: dVols present tenants with an interface closely resembling a physical device and so avoid restricting application semantics. dVols also offer tenants the ability to offload functionality not directly supported by SCMs. For example, dVols support atomic updates [26, 13] and compare-and-swap [67] operations.

Support for Operational Tasks: Decoupling storage from the underlying hardware provides a valuable point of indirection in support of datacenter resource management. In virtualizing physical storage, dVols support operational tasks such as non-disruptive migration and provide a primitive for dynamic resource schedulers to optimize the placement of dVols.

Visibility of Failure Domains: Unlike traditional vol-

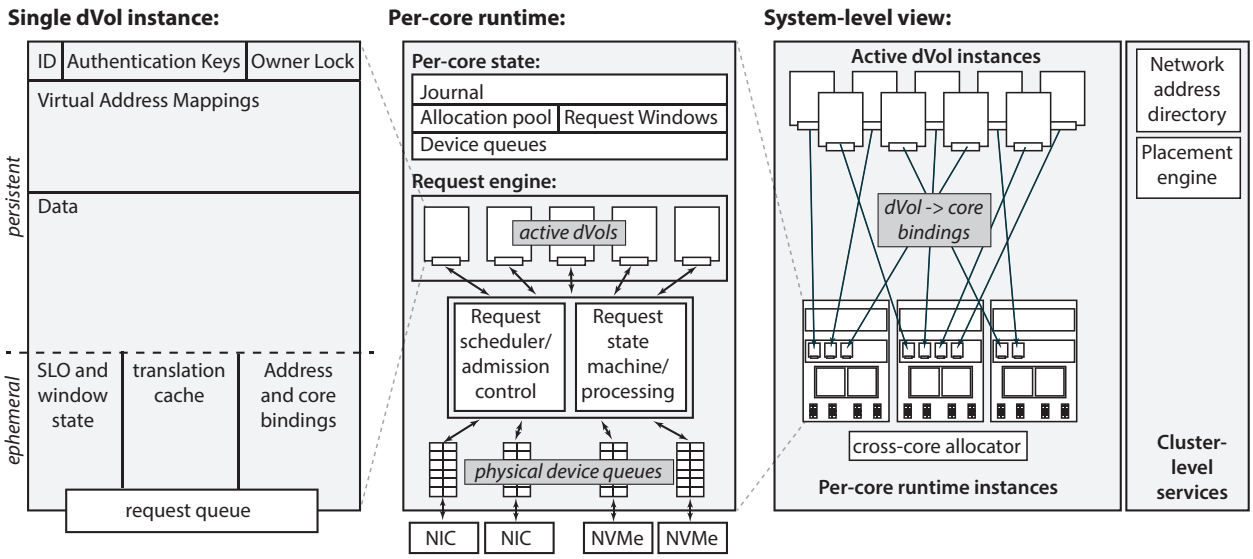

Figure 1: dVol and per-core runtime architecture in Decibel

umes that abstract information about the underlying hardware away, dVols retain and present enough device information to allow applications to reason about failure domains and to appropriately manage placement across hosts and devices.

Elastic Capacity: SCMs are arbitrarily partitioned into independent, non-contiguous dVols at runtime and, subject to device capacity constraints, can grow and shrink during execution without causing device fragmentation and a corresponding wastage of space.

Strong Isolation and Access Control: As multi-tenant storage runs the risk of performance interference due to co-location, dVols allow tenants to specify service-level objectives (SLOs) and provide cross-tenant isolation and the throughput guarantees necessary for data stores. In addition, dVols include access control mechanisms to allow tenants to specify restrictions and safeguard against unauthorized accesses and information disclosure.

As the basis for a scalable cloud storage service, Decibel represents a point in the design space that is between host-local ephemeral disks on one hand, and large-scale block storage services such as Amazon's Elastic Block Store (EBS) on the other. Like EBS volumes, dVols are separate from the virtual machines that access them, may be remapped in the face of failure, and allow a greater degree of utilization of storage resources than direct access to local disks. However, unlike volumes in a distributed block store, each dVol in Decibel is stored entirely on a single physical device, provides no storage-level redundancy, and exposes failures directly to the client.

3 The Decibel Runtime

Decibel virtualizes the hardware into dVols and multiplexes these dVols onto available physical resources to ensure isolation and to meet their performance objectives. Each Decibel instance is a single-host runtime that is responsible solely for abstracting the remote, shared nature of disaggregated storage from tenants. Decibel can provide ephemeral storage directly to tenants or act as a building block for a larger distributed storage system where multiple Decibel instances are combined to form a network filesystem or an object store [65, 15].

Decibel's architecture is shown in Figure 1: it partitions system hardware into independent, shared-nothing per-core runtimes. As achieving efficient resource utilization requires concurrent, wait-free processing of requests and needs to eliminate synchronization and coherence traffic that is detrimental to performance, Decibel opts for full, top-to-bottom system partitioning. Each per-core runtime has exclusive access to a single hardware queue for every NIC and SCM in the system and can access the hardware without requiring co-ordination across cores.

Decibel relies on kernel-bypass libraries to partition the system and processes I/O traffic within the application itself; a network stack on top of the DPDK [30] and a block layer on top of the SPDK [32] provide direct access to network and storage resources. Each per-core runtime operates independently and is uniquely addressable across the network. Execution on each core is through a single kernel thread on which the runtime co-operative schedules network and storage I/O and request processing along with virtualization and other datapath services.

vol_read *(vol, addr, len)* →*data*
vol_read_ex *(vol, addr, len)* →*(data, meta)*
vol_write *(vol, addr, len, data)* →*status*
vol_write_ex *(vol, addr, len, data, meta)* →*status*
vol_deallocate *(vol, addr[], nchunks)* →*status*
vol_write_tx *(vol, addr, len, data)* →*status*
vol_cmpxchg *(vol, addr, old, new)* →*status*

Figure 2: Datapath Interfaces for dVols. The last two provide functionality not directly provided by SCMs in hardware.

Each per-core runtime services requests from multiple tenants for multiple dVols, while each dVol is bound to a single core. The mapping from dVols to the host and core is reflected in the network address directory, which is a separate, global control path network service. As self-contained entities, dVols can be migrated across cores and devices within a host or across hosts in response to changes in load or performance objectives.

By forcing all operations for a dVol to be executed serially on a single core, Decibel avoids the contention overheads that have plagued high-performance concurrent systems [8, 10]. Binding dVols to a single core and SCM restricts the performance of the dVol to that of a single core and device, forcing Decibel to rely on client-side aggregation where higher throughput or greater capacity are required. We anticipate that the runtime can be extended to split existing hot dVols across multiple cores [2] to provide better performance elasticity.

Clients provision and access dVols using a client-side library that maps client interfaces to remote RPCs. The library also handles interaction with the network address directory, allowing applications to remain oblivious to the remote nature of dVols. Legacy applications could be supported through a network block device driver; however, this functionality is currently not provided.

3.1 Virtual Block Devices

Storage virtualization balances the need to preserve the illusion of exclusive, locally-attached disks for tenants with the necessity of supporting operational and management tasks for datacenter operators. The tenant interfaces to virtualized storage, enumerated in Figure 2, closely match that of the underlying hardware with commands such as read, write, and deallocate[2] providing the same semantics as the corresponding NVMe commands.

Device Partitioning: dVols provide tenants a sparse virtual address space backed by an SCM. As the storage requirements of tenants vary over time, the capacity utilization of dVols constantly grows and shrinks during execution. Consequently, Decibel must manage the fine-grained allocation of capacity resources across dVols.

One alternative for device partitioning is to rely on hardware-based NVMe namespaces [71] which divide SCMs into virtual partitions that may be presented directly to tenants. As implemented in modern hardware, namespaces represent large contiguous physical regions of the device, making them unsuitable for dynamically resizing workloads. Moreover, many NVMe devices do not support namespaces at all, and where they are supported, devices are typically limited to a very small number[3] of namespace instances. While the namespace idea is in principle an excellent abstraction at the device layer, these limits make them insufficient today, and are one of the reasons that Decibel elects to virtualize the SCM address space above the device itself.

Decibel virtualizes SCMs at block-granularity. Blocks are 4K contiguous regions of the device's physical address space. While some SCMs support variable block sizes, Decibel uses 4K blocks to match both existing storage system designs and x86 memory pages. Blocks are the smallest writeable unit that do not require firmware read-modify-write (RMW) cycles during updates, and also generally the largest unit that can be atomically overwritten by SCMs for crash-safe in-place updates.

Address Virtualization: dVols map the virtual address space presented to tenants onto physical blocks using a private *virtual-to-physical* (V2P) table. Each dVol's V2P table is structured as a persistent B+ tree, with fixed, block-sized internal and leaf nodes, and is keyed by 64-bit virtual addresses; internal nodes store references to their children as 64-bit physical block addresses.

V2P mappings are stored as metadata on the SCM. Client writes are fully persisted, including both data and V2P mappings, before being acknowledged. Decibel performs soft-update-ordered [19] writes of data blocks and metadata: where a write requires an update to the V2P table, data is always written and acknowledged by the device before the associated metadata write is issued. The current implementation is conservative, in that all V2P transactions are isolated. There is opportunity to further improve performance by merging V2P updates. Subsequent writes to allocated blocks do not modify the V2P table and occur in-place, relying on the block-level write atomicity of SCMs for consistency.

Modern SCMs show little benefit for physically contiguous accesses, especially in multi-tenant scenarios with

[2]We use the NVMe "deallocate" command, also termed "trim", "unmap", or "discard" in the context of SATA/SAS SSDs.

[3]The maximum number supported today is 16, with vendors indicated that devices supporting 128 namespaces are likely to be available over the next few years.

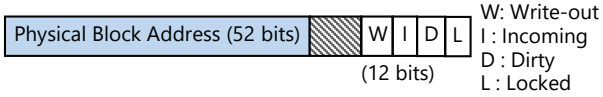

Figure 3: Cached V2P Entry

mixed reads and writes. As a result, dVols do not preserve contiguity from tenant writes and split large, variable-sized requests into multiple, block-sized ones. V2P mappings are also stored at a fixed, block-sized granularity. This trades request amplification and an increase in V2P entries for simplified system design: Decibel does not require background compaction and defragmentation services, while V2P entries avoid additional metadata for variable-sized mappings.

Decibel aggressively caches the mappings for every dVol in DRAM. The V2P table for a fully-allocated terabyte-sized device occupies approximately 6GB (an overhead of 0.6%). While non-trivial, this is well within the limits of a high performance server. Cached V2P entries vary from the on-device format: as Figure 3 illustrates, physical addresses are block-aligned and require only 52 bits, so the remaining 12 bits are used for entry metadata.

The *Incoming* and *Write-out* bits are used for cache management and signify that the entry is either waiting to be loaded from the SCM or that an updated entry is being flushed to the SCM and is awaiting an acknowledgement for the write. The *Dirty* bit indicates that the underlying data block has been modified and is used to track dirtied blocks to copy during dVol migrations. The *Locked* bit provides mutual exclusion between requests to overlapping regions of the dVol: when set, it restricts all access to the mapping for any request context besides the one that has taken ownership of the lock.

Block Allocation: Requests to allocate blocks require a consistent view of allocations across the entire system to prevent races and double allocations. Decibel amortizes the synchronization overhead of allocations by splitting them into *reservations* and *assignment*: each core reserves a fixed-size, physically-contiguous collection of blocks called an *extent* from the device in a single operation and adds it to a per-core allocation pool (resembling the thread cache in `tcmalloc`).

As seen in Figure 4, SCMs are divided into multiple extents, which are dynamically reserved by cores. The reservation of extents to cores is tracked by a global, per-device allocator. Cores asynchronously request more extents from the allocator once the number of available blocks in their local pool falls below a certain threshold. This ensures that, as long as the device is not full, allocations succeed without requiring any synchronization.

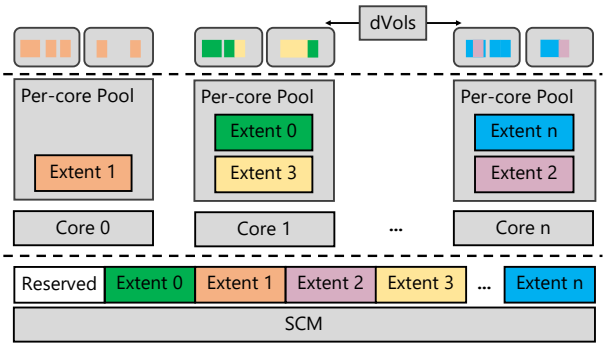

Figure 4: Physical Partitioning of Storage Devices

Assigning entire extents to dVols risks fragmentation and a wastage of space. Instead, cores satisfy allocation requests from dVols by assigning them blocks from any extent in their private pool at a single block granularity. As individual blocks from extents are assigned to different dVols, the split-allocation scheme eliminates both fragmentation and contention along the datapath.

Internally, extents track the allocation status of blocks using a single block-sized bitmap; as every bit in the bitmap represents a block, each extent is 128 MB $(4K \times 4K \times 8)$. Restricting the size of extents to the representation capacity of a single block-sized bitmap allows the bitmap to atomically be overwritten after allocations and frees.

dVols explicitly free blocks that are no longer need using the deallocate command, while deleting dVols or migrating them across devices implicitly triggers block deallocations. Freed blocks are returned to the local pool of the core where they were originally allocated and are used to fulfil subsequent allocation requests.

Data Integrity and Paravirtual Interfaces: SCMs are increasingly prone to block errors and data corruption as they age and approach the endurance limits of flash cells [57]. Even in the absence of hardware failures, SCMs risk data corruption due to *write-tearing*: since most SCMs do not support atomic multi-block updates, failures during these updates result in partial writes that leave blocks in an inconsistent state.

Decibel provides additional services not directly available in hardware to help prevent and detect data corruption. On each write, it calculates block-level checksums and verifies them on reads to detect corrupted blocks before they propagate through the system. dVols also support two additional I/O commands: *multi-block atomicity* to protect against write-tearing and *block compare-and-swap* to allow applications that can only communicate over shared storage to synchronize operations using persistent, on-disk locks [67].

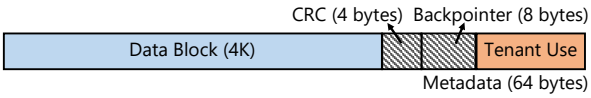

Figure 5: Extended Logical Blocks (Block + Metadata)

Several enterprise SCMs support storing per-block metadata alongside data blocks and updating the metadata atomically with writes to the data. The block and metadata regions together are called *extended logical blocks* (shown in Figure 5). Block metadata corresponds to the Data Integrity Field (DIF) provided by SCSI devices and is intended for use by the storage system. Decibel utilizes this region to store a CRC32-checksum of every block.

Block checksums are self-referential integrity checks that protect against data corruption, but offer no guarantees about metadata integrity, as V2P entries pointing to stale or incorrect data blocks are not detected. Metadata integrity can be ensured either by storing checksums with the metadata or by storing backpointers alongside the data. To avoid updating the mappings on every write, Decibel stores backpointers in the metadata region of a block. As the data, checksum, and backpointer are updated atomically, Decibel overwrites blocks in-place and still remains crash consistent.

The metadata region is exposed to tenants through the extended, metadata-aware read and write commands (see Figure 2) and can be used to store application-specific data, such as version numbers and cryptographic capabilities. As this region is shared between Decibel and tenants, the extended read and write functions mask out the checksum and backpointer before exposing the remainder of the metadata to tenants.

Since most SCMs are unable to guarantee atomicity for writes spanning multiple blocks, Decibel provides atomic updates using block-level copy-on-write semantics. First, new physical blocks are allocated and the data written, following which the corresponding V2P entries are modified to point to the new blocks. Once the updated mappings are persisted, the old blocks are freed. As the V2P entries being updated may span multiple B-tree nodes, a lightweight journal is used to transactionalize the update and ensure crash-consistency.

To perform block-level CAS operations, Decibel first ensures that there are no in-flight requests for the desired block before locking its V2P entry to prevent access until the operation is complete. The entire block is then read into memory and tested against the compare value; if they match, the swap value is written to the block. Storage systems have typically used CAS operations, when available, to co-ordinate accesses to ranges of a shared device or volume without locking the entire device.

vol_create *() →(vol, token)*
vol_restrict *(vol, type, param) →status*
vol_open *(vol, token) →status*
vol_change_auth *(vol, token) →newtoken*
vol_delete *(vol, token) →status*

Figure 6: Provisioning and Access Control Interfaces

Provisioning and Access Control: Access control mechanisms restrict a tenant's view of storage to only the dVols it is permitted to access. Decibel uses a lightweight, token-based authentication scheme for authorization and *does not* protect data confidentiality via encryption, as such CPU-intensive facilities are best left to either the clients or the hardware.

Figure 6 enumerates the control plane interfaces, presented to tenants, to provision dVols and manage access control policies for them. While creating dVols, Decibel generates a random globally unique identifier and token pair, which are returned to the tenant for use as a volume handle and credentials for future access.

In addition to credential-based authorization, dVols can also restrict access on the basis of network parameters, such as a specific IP address, or to certain VLANs and VXLANs[4]. By forcing all traffic for a dVol onto a private network segment, Decibel allows policies to be applied within the network; for example, traffic can be routed through middleboxes for packet inspection or rely on traffic shaping to prioritize latency-sensitive workloads.

3.2 Scheduling Storage

Virtualization allows tenants to operate as if deployed on private, local storage, while still benefiting from the flexibility and economic benefits of device consolidation within the datacenter. For practical deployments, preserving only an interface resembling local storage is insufficient: the storage system must also preserve the performance of the device and insulate tenants from interference due to resource contention and sharing.

The need for performance isolation in multi-tenant storage systems has led to the development of several algorithms and policies to provide fair sharing of devices and guaranteeing tenant throughput [44, 21, 68, 22, 23, 37, 61, 69, 66, 60, 73] and for providing hard deadlines for requests [37, 22, 69, 73]. Rather than prescribing a particular policy, Decibel provides dVols as a policy enforcement tool for performance isolation.

[4]Virtual Extensible LANs (VXLANs) provide support for L2-over-L4 packet tunneling and are used to build private overlay networks.

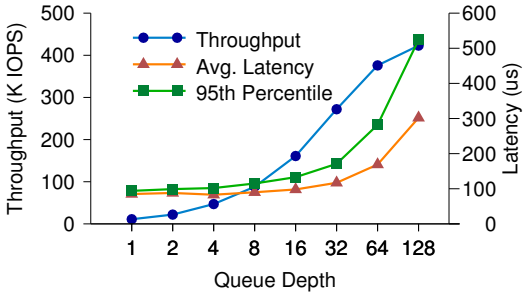

Figure 7: Throughput and Latency of Reads

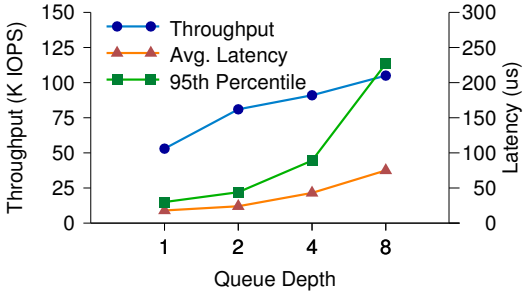

Figure 8: Throughput and Latency of Writes

SLOs: Throughput and Latency: Figures 7 and 8 compare the throughput and latency for both reads and writes of a single device, measured locally at different queue depths. The results lead us to two observations about performance isolation for SCMs:

There is a single latency class for the entire device. Even for multi-queue devices, request latency depends on overall device load. Despite the fact that the NVMe specification details multiple scheduling policies across submission queues for QoS purposes, current devices do not sufficiently insulate requests from different queues to support multiple latency classes for a single device. Instead Decibel allows the storage administrator to pick a throughput target for every SCM, called the *device ceiling*, to match a desired latency target.

Providing hard latency guarantees is not possible on today's devices. Comparing average and 95*th* percentile latencies for the device, even at relatively low utilization levels, reveal significant jitter, particularly in the case of writes. Long tail latencies have also been observed for these devices in real deployments [25]. This is largely due to the Flash Translation Layer (FTL), a hardware indirection layer that provides background bookkeeping operations such as wear levelling and garbage collection.

Emerging hardware that provides predictable performance by managing flash bookkeeping in software is discussed in Section 5. In the absence of predictable SCMs, Decibel focuses on preserving device throughput. dVols encapsulate an SLO describing their performance requirements, either in terms of a proportional share of the device or a precise throughput target.

Characterizing Request Cost: Guaranteeing throughput requires the scheduler to be able to account for the cost of every request before deciding whether to issue it to the device. Request costs, however, are variable and a function of the size and nature of the request, as well as the current load on the SCM. For example, writes are significantly cheaper than reads as long as they are being absorbed by the SCM write buffer, but become much more expensive once the write buffer is exhausted.

The Decibel scheduler does not need to account for variable-sized tenant requests as the address translation layer of the dVol converts them into uniform 4K requests at the block layer. As a simplifying assumption, the scheduler does not try and quantify the relative costs of reads and writes, but instead requires both the device ceiling and SLOs to specify read and write targets separately. In the future, we intend to extend the scheduler to provide a unified cost model for reads and writes [60].

Request Windows: At a specified device ceiling, Decibel determines the size of the global request window for an SCM, or the total number of outstanding requests that can be issued against the device. Each per-core runtime has a private request window, where the sizes of all the individual request windows are equal to that of the global request window for the device. The size of the private request window for cores is calculated on the basis of the SLO requirements of dVols scheduled to execute on that core. As dVols are created, opened, moved, or destroyed, Decibel recalculates the private window sizes, which are periodically fetched by the cores.

dVols submit requests to devices by enqueuing them in private software queues. While submitting requests to the device, the per-core runtime selects requests from the individual dVol queues until either the request window is full or there are no pending requests awaiting submission. The runtime chooses requests from multiple dVols on the basis of several factors, such as the scheduling policy, the dVol's performance requirements, and how many requests the dVols have submitted recently.

Execution Model: Per-core runtimes co-operatively schedule dVols on a single OS thread: the request processor issues asynchronous versions of blocking syscalls and yields in a timely manner. Decibel polls NICs and SCMs on the same thread to eliminate context switching overheads and to allow the schedulers to precisely con-

trol the distribution of compute cycles between servicing the hardware and the request processing within dVols.

Request processing within the dVol includes resolving virtualization mappings and performing access control checks; consequently, requests may block and yield several times during execution and cannot be run to completion as in many memory-backed systems. The scheduler treats requests and dVols as analagous to threads and processes – scheduling operates at the request level on the basis of policies applying to the entire dVol. At any given time the scheduler dynamically selects between executing the network or storage stacks, and processing one of several executable requests from multiple dVols.

Storage workloads are bursty and susceptible to incast [54]; as a result, Decibel is periodically subject to bursts of heavy traffic. At these times, the Decibel scheduler elastically steals cycles to prioritize handling network traffic. It polls NICs at increased frequencies to ensure that packets are not dropped due to insufficient hardware buffering, and processes incoming packets just enough to generate ACKs and prevent retransmissions.

Prioritizing network I/O at the cost of request processing may cause memory pressure due to an increase in number of pending requests. At a certain threshold, Decibel switches back to processing requests, even at the cost of dropped packets. As dropped packets are interpreted as network congestion, they force the sender to back-off, thus inducing back pressure in the system.

Scheduling Policies: The scheduling policy determines how the per-core runtime selects requests from multiple dVols to submit to the device. To demonstrate the policy-agnostic nature of Decibel's architecture, we prototype two different scheduling policies for dVols.

Strict Time Sharing (STS) emulates local storage by statically partitioning and assigning resources to tenants. It sacrifices elasticity and the ability to handle bursts for more predictable request latency. Each dVol is assigned a fixed request quota per scheduling epoch from which the scheduler selects requests to submit to the device. dVols cannot exceed their quota even in the absence of any competition. Further, dVols do not gain any credits during periods of low activity, as unused quota slots are not carried forward.

Deficit Round Robin (DRR) [59] is a work conserving scheduler that supports bursty tenant access patterns. DRR guarantees that each dVol is able to issue its fair share of requests to the device, but does not limit a dVol to only its fair share in the absence of competing dVols. Each dVol has an assigned quota per scheduling epoch; however, dVols that do not use their entire quota carry it forward for future epochs. As dVols can exceed their quota in the absence of competition, bursty workloads can be accommodated.

By default, Decibel is configured with DRR to preserve the flexibility benefits of disaggregated storage. This remains a configurable parameter to allow storage administrators to pick the appropriate policies for their tenants.

3.3 Placement and Discovery

Decibel makes the decision to explicitly decouple *scheduling* from the *placement* of dVols on the appropriate cores and hosts in the cluster. This division of responsibilities allows the scheduler to focus exclusively on, as seen earlier, providing fine-grained performance isolation and predictable performance over microsecond timeframes on a per-core basis. Placement decisions are made with a view of the cluster over longer timeframes in response to the changing capacity and performance requirements of dVols. Decibel defers placement decisions to an external placement engine called Mirador [70]. Mirador is a global controller that provides continuous and dynamic improvements to dVol placements by migrating them across cores, devices, and hosts and plays a role analagous to that of an SDN controller for network flows.

Storage workloads are impacted by more than just the local device; for example, network and PCIe bandwidth oversubscription can significantly impact tenant performance. Dynamic placement with global resource visibility is a response to not just changing tenant requirements, but also to connectivity bottlenecks within the datacenter. Dynamic dVol migrations, however, raise questions about how tenants locate and access their dVols.

dVol Discovery: Decibel implements a global directory service that maps dVol identifiers to the precise host and core on which they run. Cores are independent network-addressable entities with a unique `<ip:port>` identifier and can directly be addressed by tenants. The demultiplexing of tenant requests to the appropriate dVol happens at the core on the basis of the dVol identifier.

dVol Migration: The placement engine triggers migrations in response to capacity or performance shortages and aims to find a placement schedule that ensures that both dVol SLOs are met and that the free capacity of the cluster is uniformly distributed across Decibel instances to allow every dVol an opportunity to grow. Migrations can be across cores on the same host or across devices within the same or different hosts.

Core migrations occur entirely within a single Decibel instance. dVols are migrated to another core within the same host without requiring any data movement. First, Decibel flushes all the device queues and waits for in-flight requests to be completed, but no new dVol requests

are admitted. The dVol metadata is then moved to the destination core and the address directory updated. The client is instructed to invalidate its directory cache and the connection is terminated; the client then connects to the new runtime instance and resumes operations.

Device migrations resemble virtual machine migration and involve a background copy of dVol data. As the dVol continues to service requests, modified data blocks are tracked using the dirty bit in the V2P table and copied to the destination. When both copies approach convergence, the client is redirected to the new destination using the same technique as core migrations and the remaining modified blocks moved in a post-copy pass.

Decibel originally intended to perform migrations without any client involvement using OpenFlow-based redirection in the network and hardware flow steering at the end-hosts. Due to the limited availability of match rules at both switches and NICs today, Decibel opts to defer this functionality to the client library.

3.4 The Network Layer

Decibel presents the dVol interface over asynchronous TCP/IP-based RPC messages. Network flows are processed using a user space networking stack that borrows the TCP state machine and structures for processing TCP flows, such as the socket and flow tables, from mTCP [34] and combines them with custom buffer management and event notification systems. Decibel offloads checksums to the hardware, but currently does not support TCP segmentation offload.

Clients discover core mappings of dVols using the network address directory. As dVols are pinned to cores which have exclusive ownership over them, tenants must direct requests to the appropriate core on the system. Modern NICs provide the ability to precisely match specific fields in the packet header with user-defined predicates and determine the destination queue for that packet on the basis of provided rules. As each core has a unique `<ip:port>`, Decibel uses such flow steering to distribute incoming requests across cores directly in hardware.

For performance reasons, Decibel extends the shared-nothing architecture into the networking layer. It borrows ideas from scalable user space network stacks [56, 34, 45, 5, 52] and partitions the socket and flow tables into local, per-core structures that can be accessed and updated without synchronization.

Memory Management: Decibel pre-allocates large per-core regions of memory for sockets, flow tables, and socket buffers from regular memory and `mbufs` for network packets from hugepages. `mbufs` are stored in lockless, per-socket send and receive ring buffers; the latter

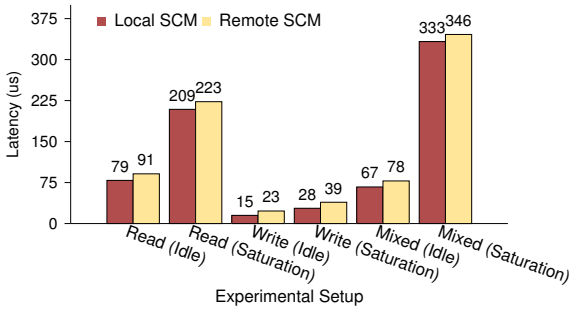

Figure 9: Local and Remote Latency for 4K Requests

is passed to DPDK which uses them to DMA incoming packets. Decibel does not support zero-copy I/O: incoming packet payloads are staged in the receive socket buffer for unpacking by the RPC layer, while writes are buffered in the send socket buffer before transmission. Zero-copy shows little benefit on processors with Direct Data I/O (DDIO), i.e., the ability to DMA directly into cache [45]. Further, once packets in `mbufs` are sent to the DPDK for transmission, they are automatically freed and unavailable for retransmissions, making zero-copy hard without complicating the programming interface.

Event Notifications and Timers: As request processing and the network stack execute on the same thread, notifications are processed in-line via callbacks. Decibel registers callbacks for new connections and for I/O events: `tcp_accept()`, `tcp_rcv_ready()`, and `tcp_snd_ready()`. Send and receive callbacks are analagous to `EPOLLIN` and `EPOLLOUT` and signify the availability of data or the ability to transmit data. The callbacks themselves do not carry any data but are only event notifications for the request processor to act on using the socket layer interfaces. Timers for flow retransmissions and connection timeouts, similarly, cannot rely on external threads or kernel interrupts to fire and instead are tracked using a hashed timer wheel and processed in-line along with other event notifications.

Why not just use RDMA? Several recent high-performance systems have exploited RDMA (Remote Direct Memory Access) – a hardware mechanism that allows direct access to remote memory without software mediation – to eliminate network overheads and construct a low-latency communication channel between servers within a datacenter, in order to accelerate network services, such as key-value stores [48, 16, 35] and data parallel frameworks [33, 24].

RDMA's advantage over traditional networking shrinks as request sizes grow [48, 35], especially in the presence of low-latency, kernel-bypass I/O libraries. Figure 9 compares local and remote access latencies, over

TCP/IP-based messaging, for SCMs when they are relatively idle (for minimum latencies) and at saturation. For a typical storage workload request size of 4K, conventional messaging adds little overhead to local accesses.

RDMA has traditionally been deployed on Infiniband and requires lossless networks for performance, making it hard to incorporate into existing Ethernet deployments. On Ethernet, RDMA requires an external control plane to guarantee packet delivery and ordering [24] and for congestion control to ensure link fairness [74].

Decibel's choice of traditional Ethernet-based messaging is pragmatic, as the advantages of RDMA for storage workloads do not yet outweigh the significant deployment overheads. As RDMA-based deployments increase in popularity, and the control plane protocols for prioritizing traffic and handling congested and lossy networks are refined, this may no longer hold true. Consequently, Decibel's architecture is mostly agnostic to the messaging layer and is capable of switching to RDMA if required by the performance of future SCMs.

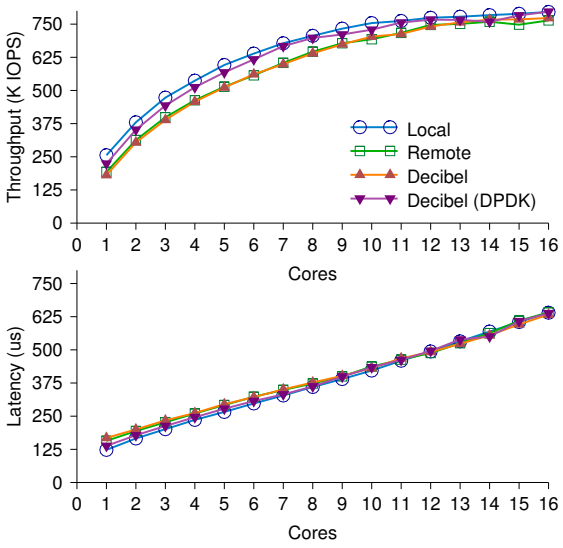

Figure 10: Performance of Decibel for a 70/30 read-write workload. Compared to local storage, Decibel has an overhead of 30µs at device saturation using a DPDK-based client.

4 Evaluation

Decibel is evaluated on a pair of 32-core Haswell systems, each with 2x40GbE X710 NICs and 4x800 GB P3700 NVMe PCIe SSDs, with one system acting as the server and the other hosting multiple clients. Each machine has 64GB RAM split across two NUMA nodes, while the 40GbE interfaces are connected via an Arista 7050 series switch. Both systems run a Linux 4.2 kernel, however, on the server Decibel takes exclusive ownership of both the network and storage adapters. Clients are measured both using the default kernel I/O stack and the DPDK-based network stack from Decibel.

At 4K request sizes, each P3700 is capable of up to 460K random read IOPS, 100K random write IOPS, and 200K random mixed (at a 70/30 read to write ratio) IOPS [31], making the saturated throughput of the system up to 1.8M read IOPS and 800K mixed IOPS. Benchmarking flash-based SSDs is non-trivial as there are a number of factors that may affect their performance. First, the performance of a new SSD is not indicative of how it would perform at *steady state* with fresh drives outperforming their steady state counterparts by a factor or two or three.

Even once steady state is reached, there is a great deal of variability in performance. Transitions from sequential to random and vice versa impact performance for several minutes, while the garbage collector can throttle disk throughput for several seconds. The P3700s, in particular, perform well past their rated write throughput for almost a minute following a period of idleness [55]. The results reported here are the average across a 10 minute run and follow industry standard guidelines for benchmarking [62]: first the devices were pre-conditioned with several weeks of heavy usage and then primed by running the same workload access pattern as the benchmark for 10 minutes prior to the benchmark run.

Remote Overhead and Scalability: Decibel is evaluated for multi-core scalability and to quantify the overhead of disaggregating SCMs when compared to direct-attached storage. All the tests are run against all 4 devices in the system, with clients evenly distributed across the cores. The clients are modelled after `fio` and access blocks randomly across the entire address space. Local clients execute as a single pinned client per-core with a queue depth of 32, while there are 2 remote clients per-core, each operating with a queue depth of 16 requests.

In the *Local* configuration, clients run directly on the server and access raw physical blocks from the SCMs without any virtualization. This local configuration serves as a baseline to compare the overhead of Decibel. In *Remote*, clients run separately from the server and request raw physical blocks across the network over TCP/IP. For *Decibel*, SCMs are virtualized into per-client dVols. Each client has a single dVol that is populated until the SCM is filled, after which they access and update blocks within the dVol. The remote configuration measures pure network overhead when compared to directly-attached SCMs, as well as the overhead of virtualization when compared to Decibel. The *Decibel (DPDK)* configuration is identical to Decibel, except that the clients bypass the kernel and use a DPDK-based network stack.

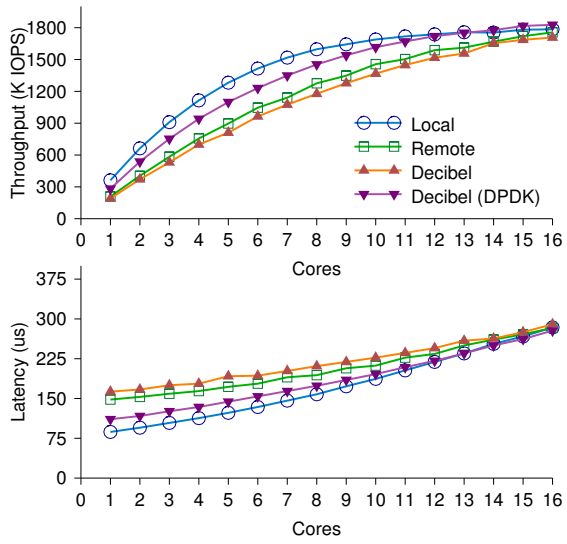

Figure 11: Performance of Decibel for an all reads workload. Compared to local storage, Decibel has an overhead of less than 20µs at device saturation using a DPDK-based client.

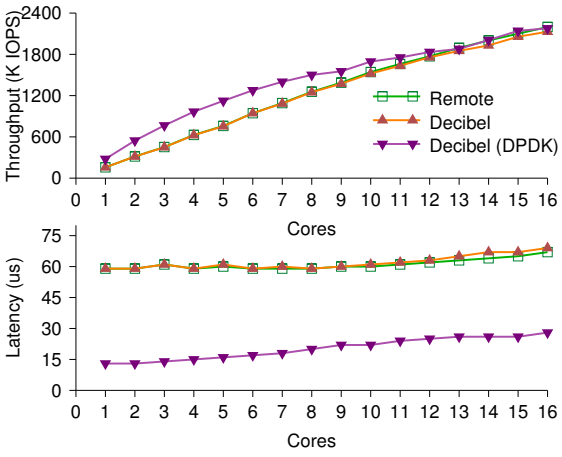

Figure 12: Remote access latencies for Decibel at different degrees of device utilization against DRAM-backed storage.

	70/30		All Reads	
	Xput	Lat	Xput	Lat
Local	750K	422	1.7M	203
Remote	740K	488	1.7M	283
Decibel	740K	490	1.7M	290
Decibel (DPDK)	750K	450	1.7M	221

Table 2: Performance for Workloads (Latency in µs)

Figure 10 compares the performance of all four configurations over 16 cores using a typical storage workload of random mixed 4K requests in a 70/30 read-write ratio. As device saturation is achieved, we do not evaluate Decibel at higher degrees of parallelism.

Decibel is highly scalable and is able to saturate all the devices, while presenting storage across the network with latencies comparable to local storage. DPDK-based clients suffer from an overhead of less than 30µs when compared to local storage, while legacy clients have overheads varying from 30-60µs depending on load.

SCMs offer substantially higher throughput for read-only workloads compared to mixed ones making them more heavily CPU-bound. Figure 11 demonstrates Decibel's ability to saturate all the devices for read-only workloads: the increased CPU load of processing requests is reflected in the gap with the local workload at low core counts. As the number of cores increase, the workload becomes SCM-bound; Decibel scales well and is able to saturate all the devices. At saturation, the DPDK-based client has an overhead of less than 20µs, while legacy clients suffer from overheads of approximately 90µs.

Once the devices are saturated, adding clients increases latency purely due to queueing delays in software. All configurations saturate the devices at less than 16 cores; hence the latency plots in Figures 10 and 11 include queueing delays and do not accurately reflect end-to-end latencies. Table 2 compares latencies at the point of device saturation: for both workloads, Decibel imposes an overhead of 20-30µs on DPDK-based clients compared to local storage.

Future SCMs, such as 3DXpoint [49], are expected to offer sub-microsecond latencies for persistent memories and approximately 10µs latencies for NVMe storage. With a view towards these devices, we evaluate Decibel against a DRAM-backed block device. As seen in Figure 12, DPDK-based clients have remote access latencies of 12-15µs at moderate load, which increases to 26µs at NIC saturation. Legacy clients have access latencies higher than 60µs, which demonstrates that the kernel stack is a poor fit for rack-scale storage architectures.

dVol Isolation: Performance isolation in Decibel is evaluated by demonstrating fair sharing of a device in two different scheduling policies when compared with a First-Come, First-Served (FCFS) scheduler that provides no performance isolation. Strict Timesharing (STS) provides static resource partitioning, while in Deficit Round Robin (DRR), dVols are prevented from interfering with the performance of other dVols, but are allowed to consume excess, unused bandwidth.

To illustrate performance isolation, Decibel is evaluated with three clients, each with a 70/30 mixed random

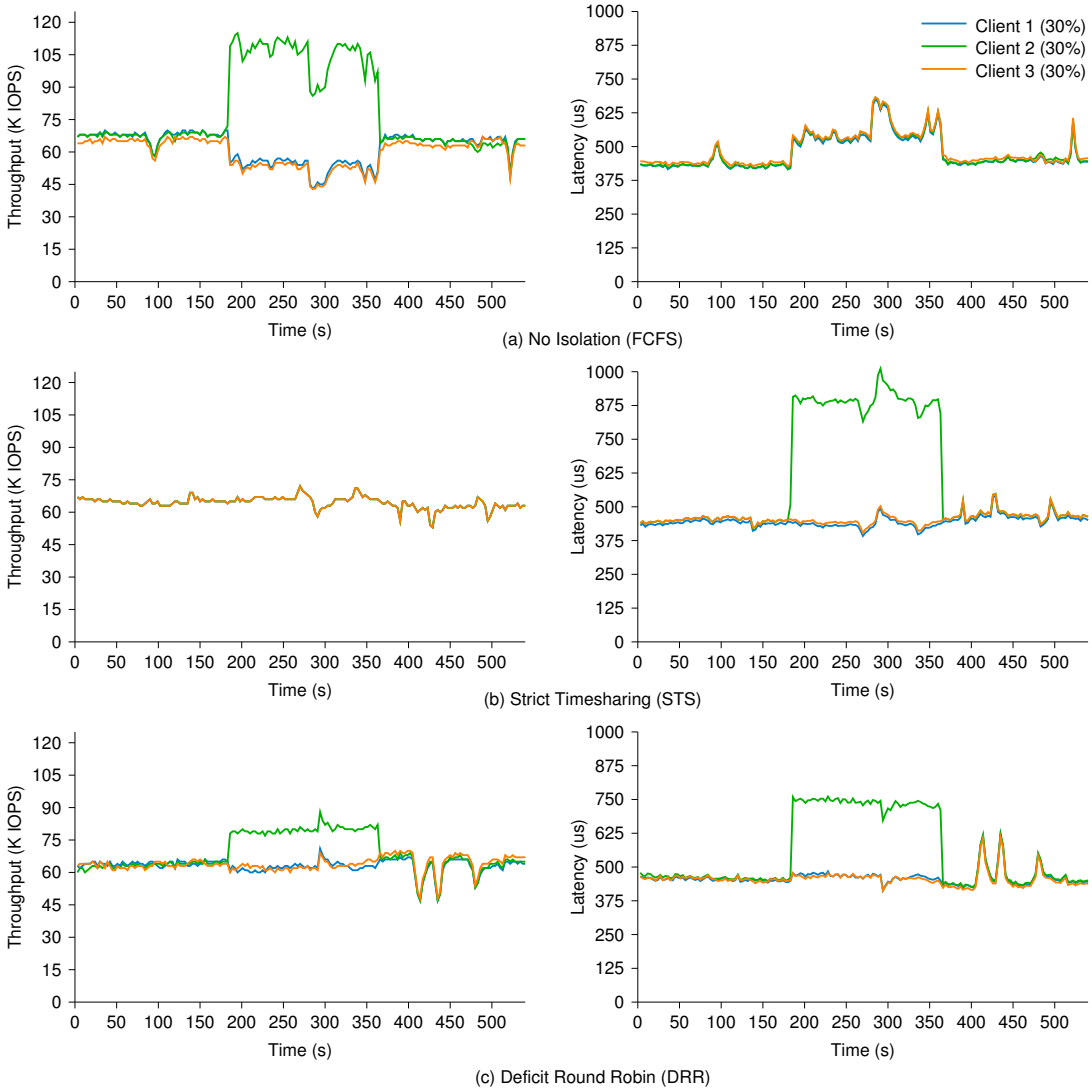

Figure 13: Isolation of a single device across multiple workloads in Decibel. Compared to the no isolation case in (a), the scheduling policies in (b) and (c), provide clients 1 and 3 a fair share of the device, even in the face of the bursty accesses of client 2.

workload, against a single shared SCM. Each client continuously issues requests to its own dVol, such that the dVol has 30 outstanding requests at any time. The dVols are configured to have an equal proportion, i.e., 30%, of the total device throughput, while the device ceiling is set to 100% utilization.

As seen in Figure 13, each client receives a throughput of 60K, leaving the device at 90% saturation. At the 3 minute mark, one of the clients experiences a traffic burst for 3 minutes such that it has 90 simultaneous in-flight requests. At 6 minutes, the burst subsides and the client returns to its original load.

FCFS offers no performance isolation, allowing the burst to create queueing overheads which impact throughput and latency of all other clients by 25%. After the burst subsides, performance returns to its original level. In contrast, STS preserves the throughput of all the clients and prevents clients from issuing any requests beyond their 30% reservation. As each dVol has hard reservations on the number of requests it can issue, requests from the bursty client are queued in software and see huge spikes in latency. The performance of the other clients remains unaffected at all times, but the excess capacity of the device remains unutilized.

DRR both guarantees the throughput of other clients and is work conserving: the bursty client consumes the unused bandwidth until the device ceiling is reached, but

not at the cost of the throughput of other clients. Latency, however, for all the clients rises slightly – this is not because of queueing delays, but because the device latency increases as it gets closer to saturation.

5 Related Work

Network-Attached Storage: The idea of centralizing storage in consolidated arrays and exporting disks over the network [38] is not a new one and has periodically been explored with changes in the relative performance of CPU, networks, and disks. For spinning disk based systems, Petal [41] is a virtualized block store that acts as a building block for a distributed file system [65]. While Petal focuses on aggregating physical disks for performance, Decibel is concerned with the performance challenges of building isolated volumes for SCMs.

More recently, network-attached storage for flash devices has been used in Corfu [4] and Strata [15]. Corfu presents a distributed log over virtualized flash block-devices while storing address translations at the clients. As the clients are trusted, co-operating entities, Corfu does not attempt to provide isolation between them. Strata focuses on providing a global address space for a scalable network file system on top of network-attached storage, and discusses the challenges in providing data plane services such as device aggregation, fault tolerance, and skew mitigation in a distributed manner. In contrast, Decibel is an example of the high-performance network-attached storage such file systems rely on, and provides the services required for multi-tenancy that cannot safely be implemented higher up the stack.

Network-attached Secure Disks (NASD) [20] explore security primitives and capabilities to allow sharing storage devices without requiring security checks at an external file manager on every request, while Snapdragon [1] uses self-describing capabilities to verify requests and limit the blocks a remote client has access to. SNAD [46] performs both tenant authentication and block encryption at the storage server to restrict unauthorized accesses.

Partitioned Data Stores: VoltDB [63] and MICA [43] are both examples of shared-nothing in-memory data stores, which explore vertical partitioning of hardware resources to allow all operations to proceed without expensive cross-core coordination. Architecturally, the per-core runtimes in Decibel resemble those in these systems with the addition of persistent storage devices and the associated datapath services.

Chronos [36] is a more general framework for partitioning applications by running several independent instances in parallel, fronted by a load balancer aware of the instance to partitioning mapping that can route requests accordingly.

Application Managed Flash: Several recent research storage systems have proposed using open-channel SSDs for more predictable performance [42, 7, 29, 58]. These devices expose internal flash channels, dies, and planes to the system and allow for application-managed software FTLs and custom bookkeeping policies. Of these systems, Flashblox has demonstrated that providing strong isolation and supporting multiple latency classes on a shared SCM requires extending full system partitioning to within the device. By binding device channels and dies directly to tenants in hardware and providing per-tenant accounting for garbage collection, it removes multiple sources of performance interference and maintains low tail latencies in the face of competing tenants.

Application-managed flash is largely complementary to Decibel and focuses largely on providing better and more flexible implementations of services currently provided by the FTL. These systems intentionally maintain a familiar block-like presentation for convenience and, as such, Decibel could integrate with such systems to provide strong end-to-end performance isolation.

6 Conclusion

SCMs represent orders of magnitude changes to the throughput, latency, and density of datacenter storage, and have caused a reconsideration in how storage is presented, managed, and accessed within modern datacenters. Decibel responds to the performance realities of SCMs by providing dVols to delegate storage to tenants within fully disaggregated storage architectures. dVols focus exclusively on virtualizing storage and isolating multiple tenants while ensuring that the storage is accompanied with a committed amount of compute and network resources to provide tenants with predictable, low-latency access to data.

7 Acknowledgments

We would like to thank our shepherd, Anirudh Badam, and the anonymous reviewers for their insightful comments and valuable feedback. Decibel has drawn on the experiences of Coho Data's engineering team in building a high-performance enterprise storage system and has especially benefited from numerous discussions with both Steven Smith and Tim Deegan. Finally, we would like to thank Dave Cohen at Intel for frequently providing a valuable "full-stack" perspective on datacenter storage.

References

[1] AGUILERA, M. K., JI, M., LILLIBRIDGE, M., MACCORMICK, J., OERTLI, E., ANDERSEN, D., BURROWS, M., MANN, T., AND THEKKATH, C. A. Block-Level Security for Network-Attached Disks. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies* (2003), FAST'03.

[2] AGUILERA, M. K., LENERS, J. B., AND WALFISH, M. Yesquel: Scalable SQL Storage for Web Applications. In *Proceedings of the 25th ACM SIGOPS Symposium on Operating Systems Principles* (2015), SOSP'15.

[3] AVAGO. ExpressFabric: Thinking Outside the Box. `http://www.avagotech.com/applications/datacenters/expressfabric`.

[4] BALAKRISHNAN, M., MALKHI, D., PRABHAKARAN, V., WOBBER, T., WEI, M., AND DAVIS, J. D. CORFU: A Shared Log Design for Flash Clusters. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (2012), NSDI'12.

[5] BELAY, A., PREKAS, G., KLIMOVIC, A., GROSSMAN, S., KOZYRAKIS, C., AND BUGNION, E. IX: A Protected Dataplane Operating System for High Throughput and Low Latency. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* (2014), OSDI'14.

[6] BJØRLING, M., AXBOE, J., NELLANS, D., AND BONNET, P. Linux Block IO: Introducing Multi-queue SSD Access on Multi-core Systems. In *Proceedings of the 6th International Systems and Storage Conference* (2013), SYSTOR'13.

[7] BJØRLING, M., GONZALEZ, J., AND BONNET, P. LightNVM: The Linux Open-Channel SSD Subsystem. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies* (2017), FAST'17.

[8] BOLOSKY, W. J., AND SCOTT, M. L. False Sharing and Its Effect on Shared Memory Performance. In *Proceedings of the 4th USENIX Symposium on Experiences with Distributed and Multiprocessor Systems* (1993), SEDMS'93.

[9] CALDER, B., WANG, J., OGUS, A., NILAKANTAN, N., SKJOLSVOLD, A., MCKELVIE, S., XU, Y., SRIVASTAV, S., WU, J., SIMITCI, H., HARIDAS, J., UDDARAJU, C., KHATRI, H., EDWARDS, A., BEDEKAR, V., MAINALI, S., ABBASI, R., AGARWAL, A., HAQ, M. F. U., HAQ, M. I. U., BHARDWAJ, D., DAYANAND, S., ADUSUMILLI, A., MCNETT, M., SANKARAN, S., MANIVANNAN, K., AND RIGAS, L. Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles* (2011), SOSP'11.

[10] CANTRILL, B., AND BONWICK, J. Real-World Concurrency. *Queue 6*, 5 (Sept. 2008).

[11] 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 ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (2012), ASPLOS'12.

[12] CAULFIELD, A. M., AND SWANSON, S. QuickSAN: A Storage Area Network for Fast, Distributed, Solid State Disks. In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (2013), ISCA'13.

[13] CONDIT, J., NIGHTINGALE, E. B., FROST, C., IPEK, E., LEE, B., BURGER, D., AND COETZEE, D. Better I/O Through Byte-addressable, Persistent Memory. In *Proceedings of the 22nd ACM SIGOPS Symposium on Operating Systems Principles* (2009), SOSP'09.

[14] CONFLUENT. Kafka Production Deployment. `http://docs.confluent.io/2.0.1/kafka/deployment.html`, 2015.

[15] CULLY, B., WIRES, J., MEYER, D., JAMIESON, K., FRASER, K., DEEGAN, T., STODDEN, D., LEFEBVRE, G., FERSTAY, D., AND WARFIELD, A. Strata: Scalable High-performance Storage on Virtualized Non-volatile Memory. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies* (2014), FAST'14.

[16] DRAGOJEVIĆ, A., NARAYANAN, D., HODSON, O., AND CASTRO, M. FaRM: Fast Remote Memory. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation* (2014), NSDI'14.

[17] EMC. DSSD D5. `https://www.emc.com/en-us/storage/flash/dssd/dssd-d5/index.htm`, 2016.

[18] FORD, D., LABELLE, F., POPOVICI, F. I., STOKELY, M., TRUONG, V.-A., BARROSO, L., GRIMES, C., AND QUINLAN, S. Availability in Globally Distributed Storage Systems. In *Pro-*

ceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (2010), OSDI'10.

[19] GANGER, G. R., MCKUSICK, M. K., SOULES, C. A. N., AND PATT, Y. N. Soft Updates: A Solution to the Metadata Update Problem in File Systems. *ACM Trans. Comput. Syst. 18*, 2 (May 2000), 127–153.

[20] GIBSON, G. A., NAGLE, D. F., AMIRI, K., BUTLER, J., CHANG, F. W., GOBIOFF, H., HARDIN, C., RIEDEL, E., ROCHBERG, D., AND ZELENKA, J. A Cost-effective, High-bandwidth Storage Architecture. In *Proceedings of the 8th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (1998), ASPLOS'98.

[21] GULATI, A., AHMAD, I., AND WALDSPURGER, C. A. PARDA: Proportional Allocation of Resources for Distributed Storage Access. In *Proccedings of the 7th USENIX Conference on File and Storage Technologies* (2009), FAST'09.

[22] GULATI, A., MERCHANT, A., AND VARMAN, P. J. pClock: An Arrival Curve Based Approach for QoS Guarantees in Shared Storage Systems. In *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (2007), SIGMETRICS'07.

[23] GULATI, A., MERCHANT, A., AND VARMAN, P. J. mClock: Handling Throughput Variability for Hypervisor IO Scheduling. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation* (2010), OSDI'10.

[24] GUO, C., WU, H., DENG, Z., SONI, G., YE, J., PADHYE, J., AND LIPSHTEYN, M. RDMA over Commodity Ethernet at Scale. In *Proceedings of the 2016 ACM SIGCOMM Conference* (2016), SIGCOMM'16.

[25] HAO, M., SOUNDARARAJAN, G., KENCHAMMANA-HOSEKOTE, D., CHIEN, A. A., AND GUNAWI, H. S. The Tail at Store: A Revelation from Millions of Hours of Disk and SSD Deployments. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies* (2016), FAST'16.

[26] HITZ, D., LAU, J., AND MALCOLM, M. File System Design for an NFS File Server Appliance. In *Proceedings of the 1994 USENIX Winter Technical Conference* (1994).

[27] HOEFLER, T., ROSS, R. B., AND ROSCOE, T. Distributing the Data Plane for Remote Storage Access. In *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems* (2015), HotOS'15.

[28] HUANG, C., SIMITCI, H., XU, Y., OGUS, A., CALDER, B., GOPALAN, P., LI, J., AND YEKHANIN, S. Erasure Coding in Windows Azure Storage. In *Proceedings of the 2012 USENIX Annual Technical Conference* (2012), ATC'12.

[29] HUANG, J., BADAM, A., CAULFIELD, L., NATH, S., SENGUPTA, S., SHARMA, B., AND QURESHI, M. K. FlashBlox: Achieving Both Performance Isolation and Uniform Lifetime for Virtualized SSDs. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies* (2017), FAST'17.

[30] INTEL. Data Plane Development Kit. `http://dpdk.org`, 2012.

[31] INTEL. Intel Solid State Drive DC P3700 Series. `http://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/ssd-dc-p3700-spec.pdf`, October 2015.

[32] INTEL. Storage Plane Development Kit. `https://01.org/spdk`, 2015.

[33] ISLAM, N. S., RAHMAN, M. W., JOSE, J., RAJACHANDRASEKAR, R., WANG, H., SUBRAMONI, H., MURTHY, C., AND PANDA, D. K. High Performance RDMA-based Design of HDFS over InfiniBand. In *Proceedings of the IEEE International Conference on High Performance Computing, Networking, Storage and Analysis* (2012), SC'12.

[34] JEONG, E. Y., WOO, S., JAMSHED, M., JEONG, H., IHM, S., HAN, D., AND PARK, K. mTCP: A Highly Scalable User-level TCP Stack for Multicore Systems. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation* (2014), NSDI'14.

[35] KALIA, A., KAMINSKY, M., AND ANDERSEN, D. G. Using RDMA Efficiently for Key-value Services. In *Proceedings of the 2014 ACM SIGCOMM Conference* (2014), SIGCOMM'14.

[36] KAPOOR, R., PORTER, G., TEWARI, M., VOELKER, G. M., AND VAHDAT, A. Chronos: Predictable Low Latency for Data Center Applications. In *Proceedings of the 3rd ACM Symposium on Cloud Computing* (2012), SoCC'12.

[37] KARLSSON, M., KARAMANOLIS, C., AND ZHU, X. Triage: Performance Differentiation for Storage Systems Using Adaptive Control. *ACM Transactions on Storage (TOS) 1*, 4 (Nov. 2005).

[38] KATZ, R. H. High Performance Network and Channel-Based Storage. Tech. Rep. UCB/CSD-91-650, EECS Department, University of California, Berkeley, Sep 1991.

[39] KLIMOVIC, A., KOZYRAKIS, C., THERESKA, E., JOHN, B., AND KUMAR, S. Flash storage disaggregation. In *Proceedings of the 11th ACM SIGOPS European Conference on Computer Systems* (2016), EuroSys'16.

[40] KUNZ, G. Impact of Shared Storage on Apache Cassandra. `http://www.datastax.com/dev/blog/impact-of-shared-storage-on-apache-cassandra`, January 2017.

[41] LEE, E. K., AND THEKKATH, C. A. Petal: Distributed Virtual Disks. In *Proceedings of the 7th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (1996), ASPLOS'96.

[42] LEE, S., LIU, M., JUN, S., XU, S., KIM, J., AND ARVIND, A. Application-managed Flash. In *Proceedings of the 14th Usenix Conference on File and Storage Technologies* (2016), FAST'16, pp. 339–353.

[43] LIM, H., HAN, D., ANDERSEN, D. G., AND KAMINSKY, M. MICA: A Holistic Approach to Fast In-memory Key-value Storage. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation* (2014), NSDI'14.

[44] LUMB, C. R., MERCHANT, A., AND ALVAREZ, G. A. Facade: Virtual Storage Devices with Performance Guarantees. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies* (2003), FAST'03.

[45] MARINOS, I., WATSON, R. N., AND HANDLEY, M. Network Stack Specialization for Performance. In *Proceedings of the 2014 ACM SIGCOMM Conference* (2014), SIGCOMM'14.

[46] MILLER, E. L., LONG, D. D. E., FREEMAN, W. E., AND REED, B. Strong Security for Network-Attached Storage. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies* (2002), FAST'02.

[47] MILLER, J. Why does my choice of storage matter with Cassandra? `http://www.slideshare.net/johnny15676/why-does-my-choiceofstorage-matterwithcassandra`, November 2014.

[48] MITCHELL, C., GENG, Y., AND LI, J. Using One-sided RDMA Reads to Build a Fast, CPU-efficient Key-value Store. In *Proceedings of the 2013 USENIX Annual Technical Conference* (2013), ATC'13.

[49] MORGAN, T. P. Intel Shows Off 3D XPoint Memory Performance. `http://www.nextplatform.com/2015/10/28/intel-shows-off-3d-xpoint-memory-performance/`, October 2015.

[50] OUYANG, J., LIN, S., JIANG, S., HOU, Z., WANG, Y., AND WANG, Y. SDF: Software-defined Flash for Web-scale Internet Storage Systems. In *Proceedings of the 19th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (2014), ASPLOS'14.

[51] PATTERSON, D. A., GIBSON, G., AND KATZ, R. H. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data* (1988), SIGMOD'88.

[52] PETER, S., LI, J., ZHANG, I., PORTS, D. R. K., WOOS, D., KRISHNAMURTHY, A., ANDERSON, T., AND ROSCOE, T. Arrakis: The Operating System is the Control Plane. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* (2014), OSDI'14.

[53] PETERSEN, C. Introducing Lightning: A flexible NVMe JBOF. `https://code.facebook.com/posts/989638804458007/introducing-lightning-a-flexible-nvme-jbof/`, March 2016.

[54] PHANISHAYEE, A., KREVAT, E., VASUDEVAN, V., ANDERSEN, D. G., GANGER, G. R., GIBSON, G. A., AND SESHAN, S. Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies* (2008), FAST'08.

[55] RILEY, D. Intel SSD DC P3700 800GB and 1.6TB Review: The Future of Storage. `http://www.tomshardware.com/reviews/intel-ssd-dc-p3700-nvme,3858-5.html`, August 2014.

[56] RIZZO, L. Netmap: A Novel Framework for Fast Packet I/O. In *Proceedings of the 2012 USENIX Annual Technical Conference* (2012), ATC'12.

[57] SCHROEDER, B., LAGISETTY, R., AND MERCHANT, A. Flash Reliability in Production: The Expected and the Unexpected. In *Proceedings of the 14th Usenix Conference on File and Storage Technologies* (2016), FAST'16.

[58] SHEN, Z., CHEN, F., JIA, Y., AND SHAO, Z. DIDACache: A Deep Integration of Device and Application for Flash-Based Key-Value Caching. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies* (2017), FAST'17.

[59] SHREEDHAR, M., AND VARGHESE, G. Efficient Fair Queueing Using Deficit Round Robin. In *Proceedings of the 1995 ACM SIGCOMM Conference* (1995), SIGCOMM'95.

[60] SHUE, D., AND FREEDMAN, M. J. From Application Requests to Virtual IOPs: Provisioned Key-value Storage with Libra. In *Proceedings of the 9th ACM SIGOPS European Conference on Computer Systems* (2014), EuroSys'14.

[61] SHUE, D., FREEDMAN, M. J., AND SHAIKH, A. Performance Isolation and Fairness for Multi-tenant Cloud Storage. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (2012), OSDI'12.

[62] SPANJER, E., AND HO, E. The Why and How of SSD Performance Benchmarking. `http://www.snia.org/sites/default/education/tutorials/2011/fall/SolidState/EstherSpanjer_The_Why_How_SSD_Performance_Benchmarking.pdf`, 2011.

[63] STONEBRAKER, M., MADDEN, S., ABADI, D. J., HARIZOPOULOS, S., HACHEM, N., AND HELLAND, P. The End of an Architectural Era: (It's Time for a Complete Rewrite). In *Proceedings of the 33rd International Conference on Very Large Data Bases* (2007), VLDB'07, pp. 1150–1160.

[64] SWANSON, S., AND CAULFIELD, A. Refactor, Reduce, Recycle: Restructuring the I/O Stack for the Future of Storage. *Computer 46*, 8 (Aug. 2013).

[65] THEKKATH, C. A., MANN, T., AND LEE, E. K. Frangipani: A Scalable Distributed File System. In *Proceedings of the 16th ACM SIGOPS Symposium on Operating Systems Principles* (1997), SOSP'97.

[66] THERESKA, E., BALLANI, H., O'SHEA, G., KARAGIANNIS, T., ROWSTRON, A., TALPEY, T., BLACK, R., AND ZHU, T. IOFlow: A Software-defined Storage Architecture. In *Proceedings of the 24th ACM SIGOPS Symposium on Operating Systems Principles* (2013), SOSP'13.

[67] VMWARE. VMware vSphere Storage APIs Array Integration (VAAI). `http://www.vmware.com/resources/techresources/10337`, December 2012.

[68] WACHS, M., ABD-EL-MALEK, M., THERESKA, E., AND GANGER, G. R. Argon: Performance Insulation for Shared Storage Servers. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies* (2007), FAST'07.

[69] WANG, A., VENKATARAMAN, S., ALSPAUGH, S., KATZ, R., AND STOICA, I. Cake: Enabling High-level SLOs on Shared Storage Systems. In *Proceedings of the 3rd ACM Symposium on Cloud Computing* (2012), SoCC'12.

[70] WIRES, J., AND WARFIELD, A. Mirador: An Active Control Plane for Datacenter Storage. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies* (2017), FAST'17.

[71] WORKGROUP, N. E. NVM Express revision 1.2 Specification. `http://nvmexpress.org/wp-content/uploads/NVM_Express_1_2_Gold_20141209.pdf`, November 2014.

[72] YANG, J., MINTURN, D. B., AND HADY, F. When Poll is Better Than Interrupt. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies* (2012), FAST'12.

[73] ZHU, T., TUMANOV, A., KOZUCH, M. A., HARCHOL-BALTER, M., AND GANGER, G. R. PriorityMeister: Tail Latency QoS for Shared Networked Storage. In *Proceedings of the 5th ACM Symposium on Cloud Computing* (2014), SOCC'14.

[74] ZHU, Y., ERAN, H., FIRESTONE, D., GUO, C., LIPSHTEYN, M., LIRON, Y., PADHYE, J., RAINDEL, S., YAHIA, M. H., AND ZHANG, M. Congestion Control for Large-Scale RDMA Deployments. In *Proceedings of the 2015 ACM SIGCOMM Conference* (2015), SIGCOMM'15.