DoCeph: DPU-Offloaded Messaging in Ceph for Reduced Host CPU Utilization

Kyuli Park
Sogang University
Seoul, Republic of Korea
kyuripark@sogang.ac.kr

Sungmin Yoon
Sogang University
Seoul, Republic of Korea
zserty12@sogang.ac.kr

Farid Talibli
Sogang University
Seoul, Republic of Korea
faridt@sogang.ac.kr

Sungyong Park
Sogang University
Seoul, Republic of Korea
parksy@sogang.ac.kr

Jae-Hyuck Kwak
Korea Institute of Science and Technology Information
Daejeon, Republic of Korea
jhkwak@kisti.re.kr

Kimoon Jeong
Korea Institute of Science and Technology Information
Daejeon, Republic of Korea
kmjeong@kisti.re.kr

Awais Khan
Oak Ridge National Laboratory
Oak Ridge, TN, USA
khana@ornl.gov

Youngjae Kim*
Sogang University
Seoul, Republic of Korea
youkim@sogang.ac.kr

Abstract

Ceph is a widely used distributed object store, but its messenger layer imposes substantial CPU overhead on the host. To address this limitation, we propose DoCeph, a DPU-offloaded storage architecture for Ceph that disaggregates the system by offloading the communication-intensive messaging component to the DPU while retaining the storage backend on the host. The DPU efficiently manages communication, using lightweight RPC for metadata operations and DMA for data transfer. Moreover, DoCeph introduces a pipelining technique that overlaps data transmission with buffer preparation, mitigating hardware-imposed transfer size limitations. We implemented DoCeph on a Ceph cluster with NVIDIA BlueField-3 DPUs. Evaluation results indicate that DoCeph cuts host CPU usage by up to 92% while sustaining stable throughput and providing larger performance benefits for object writes over 1 MB.

CCS Concepts

• **Computer systems organization → Architectures**.

Keywords

Distributed Object Storage System, Ceph, Disaggregated Storage, Data Processing Unit (DPU) Offloading,

ACM Reference Format:
Kyuli Park, Sungmin Yoon, Farid Talibli, Sungyong Park, Jae-Hyuck Kwak, Kimoon Jeong, Awais Khan, and Youngjae Kim. 2025. DoCeph: DPU-Offloaded Messaging in Ceph for Reduced Host CPU Utilization. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3731599.3767525

*Y. Kim is the corresponding author.

SC Workshops '25, St Louis, MO, USA

ACM ISBN 979-8-4007-1871-7/2025/11
https://doi.org/10.1145/3731599.3767525

1 Introduction

Ceph [22] is a widely adopted open-source distributed storage system that provides an object-based design, strong scalability, and self-healing capabilities. It supports block, file, and object interfaces, making it suitable for diverse applications. The core component of Ceph, the Object Storage Daemon (OSD), is responsible for data storage, replication, and consistency. In large-scale clusters, OSDs frequently exchange messages to coordinate replication, recovery, and rebalancing.

These communications are managed by Ceph's messenger layer, which asynchronously handles heartbeats, replication traffic, and control messages at regular intervals. While this mechanism ensures consistency and balance across the cluster, it imposes heavy computational overhead. Network stack traversal, data serialization, TCP/IP transmission, compression, checksumming, and encryption are all executed by the host CPU [20].

To mitigate such overhead, Data Processing Units (DPUs) have emerged as a promising solution for data-intensive systems [1, 2, 5, 8, 14, 16, 18]. A DPU integrates ARM-based CPU cores, an independent operating system, a network interface controller (NIC), and hardware accelerators into a single system-on-chip (SoC). This architecture enables in-path data processing without host CPU involvement, making DPUs an attractive platform for offloading communication-intensive operations in storage systems.

Recent research has demonstrated the versatility of DPUs by exploring in-path read processing in storage systems [25], GPU-direct object storage [9], hardware-accelerated compression and erasure coding [12, 19], and flush offloading in key-value stores [6]. These efforts highlight the potential of DPUs for accelerating specific data-path operations. Nonetheless, the fundamental challenge of reducing host CPU overhead in distributed storage systems like Ceph remains largely unaddressed. A key difficulty lies in the fact that core networking and storage functions are tightly coupled

with cluster consistency mechanisms, and fully offloading them to external devices risks compromising reliability and fault tolerance.

In this paper, we propose DoCeph, a DPU-offloaded Ceph architecture that physically decouples OSD logic from the object store. DoCeph executes core OSD functionalities—including client request handling and message processing—on the DPU's ARM cores, while the host is dedicated to managing the backend object store (BlueStore). This separation relieves the host CPU from Ceph's network operations, which typically involve frequent memory copies and context switches through the TCP/IP kernel stack. By offloading these tasks to the DPU's TCP/IP stack, the host CPU is freed to focus exclusively on backend storage operations with substantially reduced load.

To connect the DPU-resident OSD with the host-resident BlueStore, we introduce ProxyObjectStore, a transparent intermediate layer. ProxyObjectStore routes small control-path requests (e.g., metadata lookups) through a lightweight RPC channel, while large data transfers are handled via high-performance DOCA DMA channels. This design bypasses the host kernel's network stack during bulk data transfers, thereby eliminating redundant CPU intervention and memory copy overhead.

In addition, DoCeph employs pipelined DMA execution and memory region caching to maximize throughput and minimize overhead by overlapping buffer preparation with data transmission, even under hardware-imposed DMA size limits (e.g., 2 MB). To enhance robustness, we further incorporate adaptive fallback and cooldown mechanisms that sustain reliable operation in the presence of transmission errors.

We implemented DoCeph on a Ceph cluster equipped with NVIDIA BlueField-3 DPUs and evaluated its performance against a conventional host-based Ceph deployment. Experimental results show that DoCeph reduces host CPU utilization by over 90% across diverse write workloads, while maintaining stable throughput and latency as I/O sizes increase.

2 Background and Motivation

2.1 Ceph Storage System

Ceph is a highly scalable distributed storage system that has been widely adopted in cloud infrastructure, telecom networks, and data-intensive research environments due to its flexible architecture and unified interface. It supports block storage (RBD), a POSIX-compatible file system (CephFS), and object storage (RGW) within a single cluster, allowing diverse applications to share a common storage backend.

Internally, Ceph relies on RADOS (Reliable Autonomic Distributed Object Store) [24] to manage object-based data distribution across the cluster. Each object is assigned to a logical unit called a Placement Group (PG), and the CRUSH (Controlled Replication Under Scalable Hashing) algorithm [23] deterministically maps each PG to a set of Object Storage Daemons (OSDs) for replication and placement.

Ceph consists of multiple daemon processes, each responsible for a specific cluster-level role. Among them, the OSD daemon integrates tightly coupled subsystems—including the network stack, object store engine, and replication logic—to handle data operations efficiently. Monitors (MONs) maintain cluster state and manage core metadata such as the OSDMap and MONMap. The Manager (MGR) daemon collects runtime statistics and provides modular extensions including dashboards and load tracking. When CephFS is used, Metadata Servers (MDSs) handle namespace operations and directory metadata. To enable coordination among these components, all communication in Ceph is managed by the messenger layer. It handles essential functions such as serving clients I/O, data replication between OSDs, coordinating recovery, and exchanging heartbeat messages for health monitoring. Ceph enables dynamic scaling without service interruption and ensures fault tolerance through automatic recovery and data rebalancing, making it suitable for mission-critical deployments.

2.2 Data Processing Unit

The Data Processing Unit (DPU) is a rapidly emerging class of processor designed to offload data-centric workloads such as networking and storage, thereby alleviating CPU bottlenecks in modern data centers. By independently handling system-level services, the DPU enhances both performance and resource efficiency. Its energy-efficient architecture and programmable design allow it to execute tasks such as packet processing, storage control, and security functions without CPU involvement. As a result, major chip vendors—including NVIDIA (BlueField) [16], Intel (IPU) [8], AMD (Pensando) [1], Marvell (Octeon) [14], and Broadcom (Stingray) [2]—have introduced various DPU products, while hyperscalers such as AWS (Nitro) [18], Alibaba (CIPU) [5], and Microsoft operate their own DPU-based infrastructure at scale [15].

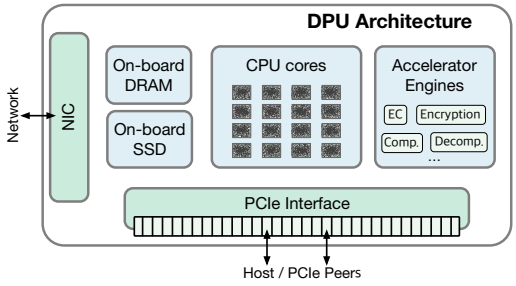

Figure 1: An architectural overview of DPU [16].

Modern DPUs are built as tightly integrated SoCs, optimized to process data-intensive operations efficiently. Figure 1 illustrates the typical architectural components of a DPU. The compute subsystem consists of multiple low-power CPU cores based on ARM or RISC-V architectures, running a standalone Linux-based operating system. These cores manage data flow, coordinate offloaded operations, and support control-plane functionality. In addition, DPUs are equipped with hardware accelerator engines that efficiently handle tasks such as erasure coding, encryption, compression, and decompression.

Each DPU includes onboard DDR memory used for buffering active datasets, as well as embedded flash storage for hosting the OS root filesystem. A PCIe interface enables the DPU to communicate with the host system and other peer devices on the PCIe fabric via DMA or peer-to-peer communication.

DPUs uniquely combine general-purpose cores, a standalone operating system, and a high-throughput NIC within a single SoC

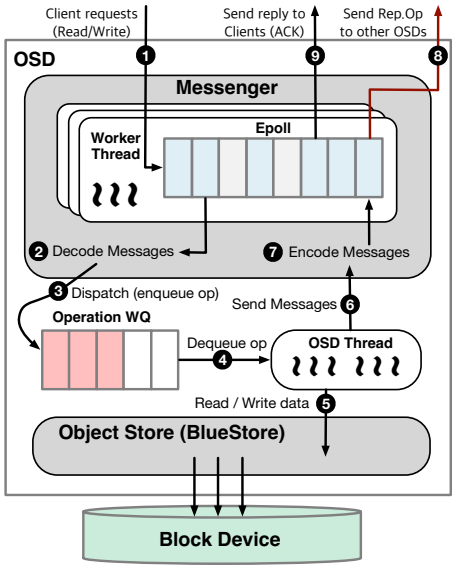

Figure 2: Internal messaging flow in Ceph OSD.

package. This integration allows them to perform in-path data processing independently of the host CPU. By integrating control logic, I/O optimization, and hardware acceleration, DPUs are emerging as a key building block for re-architecting modern data center infrastructure.

2.3 Motivation for Offloading Ceph Messenger

In Ceph, each OSD runs its own messenger instance with a pool of worker threads, where each thread handles socket connections asynchronously through a single-threaded epoll-based event loop.

Figure 2 illustrates the typical processing flow within an OSD — from receiving a client request to sending a response after completing replication. When a client read or write request arrives via a TCP socket ①, a worker thread blocked on epoll is triggered to read and decode the message ②. The decoded operation is then dispatched to the operation work queue ③, where an OSD thread picks it up ④ and performs the actual I/O through the ObjectStore (e.g., BlueStore) ⑤. The OSD thread generates a replication message, which is passed back to the messenger ⑥. The messenger encodes the message ⑦ and transmits it to secondary OSDs ⑧. Once all acknowledgments from the replicas are received, a final response is sent back to the client ⑨.

The intensive communication handled by the messenger layer imposes significant CPU overhead. The messenger repeatedly invokes system calls for TCP communication, triggering frequent context switches and buffer copies for each I/O request, while also performing message encoding and decoding operations. Our measurements in Section 5.2 show that the messenger consistently accounts for over 80% of total Ceph CPU usage across different network configurations (1 Gbps and 100 Gbps), and generates nearly an order of magnitude more context switches than ObjectStore operations.

This concentration of CPU usage in the messenger not only wastes costly host CPU cycles but also undermines energy efficiency. To address this, we leverage ARM-based DPUs with an integrated TCP/IP stack to offload network-intensive processing from the messenger layer, significantly reducing host CPU usage. This offloading frees host resources for backend I/O tasks, enabling a more energy-efficient and cost-effective disaggregated storage system.

3 Design of DoCeph

3.1 Overall Architecture

The architecture of DoCeph consists of two main components:

- (1) the DPU, which executes the full Ceph OSD stack—including replication, data placement, and client request handling
- (2) the host, which runs only a BlueStore server responsible for physical storage management.

To enable data transfer between the two components, we introduce a transparent layer on the DPU called `ProxyObjectStore`. This layer mediates all interactions between the DPU and the host through a `Proxy Interface`, which runs on both sides. Ceph's architecture defines the `ObjectStore` as a pluggable backend interface, allowing different storage technologies (e.g., BlueStore [21], FileStore [3]) to serve as the OSD's backend. DoCeph leverages this modularity by overriding the `ObjectStore` interface on the DPU with a custom proxy (`ProxyObjectStore`) that forwards backend calls to the BlueStore instance running on the host. Figure 3 illustrates the overall architecture of DoCeph.

3.2 Communication Management

Backend requests issued by DPU-resident OSD threads first enter `ProxyObjectStore`, which selects the transport based on operation type. Control-plane operations (e.g., `stat`, `exists`) are served via lightweight RPCs, whereas data-plane operations that move bulk I/O (e.g., `queue_transactions`) are staged in memory and then transferred using high-throughput DMA.

For write requests, all data-plane operations are staged in DPU memory before being sent to the host. This design enables pipelining of network reception, DMA transfer, and host BlueStore commit, thereby overlapping these phases to maximize throughput. Client acknowledgments are issued only after the host-side BlueStore completes the commit, preserving Ceph's write-through consistency semantics and crash resilience. For each request, the proxy interface in `ProxyObjectStore` serializes the message and forwards it to a lightweight host-side server, which deserializes the payload and executes the corresponding BlueStore operation.

The ProxyObjectStore is responsible for selecting the communication channel based on the operation type. This selection follows a clear binary classification: all client I/O operations are directed to the data plane for maximum throughput, while metadata management and cluster coordination operations use the control plane.

Read requests follow a similar architectural pattern: when clients issue read requests, the DPU uses ProxyObjectStore to send the necessary metadata to the host via DMA, and after the host performs the read operation, the requested data is transferred back to the DPU via DMA before being sent to the client. This bidirectional approach maintains consistent proxy interface semantics and performance characteristics for both read and write operations.

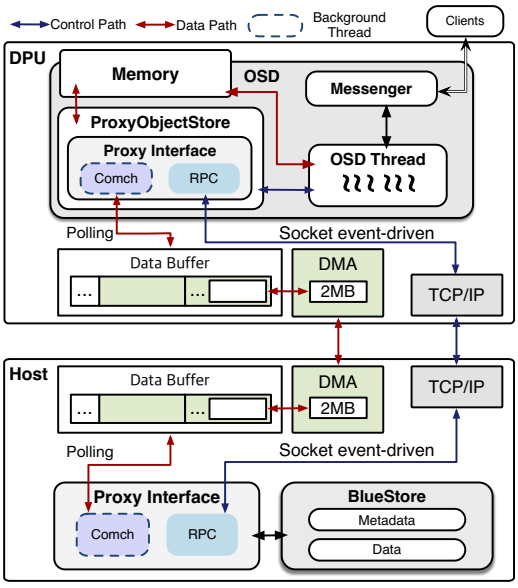

Figure 3: An architectural view of DoCeph.

Control-Plane Operations: Lightweight tasks such as metadata management or status monitoring have minimal impact on overall performance. They primarily serve to coordinate and manage cluster state rather than handle bulk data movement. A standard TCP/IP socket connection is used for the requests that are not performance-critical. The serialized RPC message is sent over a persistent TCP or UNIX domain socket. This provides a reliable and straightforward communication channel for exchanging small, infrequent messages. This channel is initialized once when the OSD starts.

Data-Plane Operations: For read and write operations that involve frequent and large-scale data transfers, we leverage NVIDIA's DOCA API to perform efficient, DMA-based direct memory transfers. The DOCA Communication Channel (CommChannel) [17] provides an API to establish and manage high-speed DMA transfer channels between the DPU and the host, exposing memory regions in advance and handling the necessary negotiation to coordinate access. The data plane operations proceed as follows.

- Upon receiving a write request from the client, the OSD uses the ProxyObjectStore to initiate a negotiation with the host via the DOCA CommChannel and allocates the memory region for DMA.
- The actual data is then directly copied from the DPU to the pre-exposed memory region on the host using the DOCA DMA engine. This process bypasses the host kernel's network stack entirely, minimizing overhead through zero-copy transfer.
- Once the DMA transfer is complete, BlueStore writes the data to the physical storage device.

A naive separation of the OSD and BlueStore components can potentially degrade performance due to additional communication overhead and synchronization delays. To mitigate this, DoCeph employs a pipelined transfer mechanism that maximizes data transfer efficiency across the end-to-end path, starting from the client, through the DPU, to the host-side DMA engine, and finally to the BlueStore storage backend.

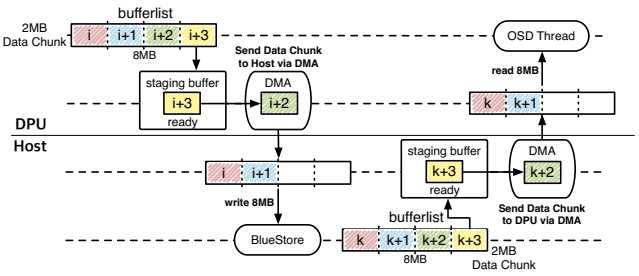

Figure 4: DMA pipelining mechanism for Host-DPU data transfer.

In DoCeph, both the DMA transmission and the preparation of incoming data buffers are managed by a dedicated background thread running in polling mode. This thread continuously monitors the DMA completion status and proactively prepares the next set of data in memory. As soon as the current DMA transfer is initiated, the system begins staging the next client request into a DMA-accessible buffer, ensuring that data preparation and data transmission can proceed concurrently.

3.3 DMA Pipelining in DoCeph

As illustrated in Figure 4, DoCeph utilizes staging buffers before DMA transmission and write buffers after transmission to pipeline the entire process. Although hardware constraints require DMA requests to be transmitted in 2MB segments, DoCeph minimizes transmission delays by reusing pre-established memory regions instead of performing CommChannel negotiation for each transfer. On the receiving end, BlueStore write operations are performed on buffered data once the complete request arrives.

This pipelining mechanism functions identically for read operations. During reads, staging buffers are positioned on the host side to facilitate the same efficient overlapped execution for DMA transfers from the host to the DPU. This architecture ensures symmetric performance optimization for bidirectional data flow between the DPU and host. Through these optimizations, DoCeph maintains a queue of pending write requests that are preloaded into available DMA-capable buffers, rather than enforcing strict sequential processing. This allows the system to overlap communication and computation phases, reducing per-request latency and minimizing idle cycles on the DPU. The resulting pipelined execution improves both throughput and responsiveness by ensuring continuous utilization of the DMA engine and memory buffers.

4 Implementation

For the Ceph OSD, running on the DPU, the ProxyObjectStore implements the standard ObjectStore interface. When a function such as `queue_transactions` or `stat` is called, the arguments are serialized (e.g., collection ID, object handles, transaction data) into a bufferlist. This bufferlist forms the payload of an RPC message. Each RPC call is encapsulated in a header containing the operation type, a unique request ID for tracking, and the payload length. This serialized message is then transmitted over the appropriate communication channel.

For control-plane messages (metadata, coordination), the ProxyObjectStore relies on a persistent socket channel. The control-plane RPC server on the host runs a persistent socket listener that monitors incoming messages. Upon receiving a message, it parses the operation type and dispatches the corresponding BlueStore handler, effectively acting as an event-driven loop. In contrast, data-plane operations such as DMA writes are handled via polling. A background thread on the host continuously polls the DOCA DMA engine. Once a DMA transfer completes, the thread immediately triggers the corresponding write handler in BlueStore, enabling zero-copy data persistence without traversing the kernel network stack.

Due to hardware limitations, a single DMA transfer is restricted to approximately 2MB [10]. To accommodate larger data transfers, DoCeph adopts a segmentation strategy that divides the request into multiple segments. When a client issues a write request exceeding the maximum DMA transfer size, the total request of size N is divided into $k = N/2MB$ segments, where each segment size is determined as the minimum of the maximum transferable size and the remaining bytes.

While this design improves throughput for typical data flows, it also has to remain resilient under failure conditions. To handle exceptional cases such as DMA transfer failures, DoCeph provides a fallback mechanism. If an error occurs during the transfer of a segment or an entire batch, the system immediately falls back to a socket-based RPC path. Previously completed segments are preserved to avoid redundant transmission. On the DPU side, an atomic cool down flag and expiration timestamp are maintained. When a failure occurs, DMA transfers are temporarily disabled for a fixed cool down period during which all requests are routed through the RPC path. After the cooldown expires, the system attempts a small test DMA transfer to determine whether the DMA path can be safely reactivated. This mechanism ensures progress in the presence of failures and enhances system robustness.

The current implementation evaluated in this paper focuses exclusively on write operations (write path), and support for read operations will be incorporated in future work. Thus, the evaluation presents preliminary results based solely on write workloads.

5 Evaluation

5.1 Experimental Setup

Our evaluation is carried out on a three-node testbed. One machine, built with a conventional CPU-only configuration, serves as the *client node* that issues benchmark traffic. The other two machines serves as *cluster nodes* running the Ceph daemons. The detailed hardware specification of each cluster node is summarized in Table 1.

Performance is compared under two configurations, `Baseline` and DoCeph, both connected through a 100Gbps Ethernet link. In Baseline, the BlueField-3 SmartNIC operates in *NIC mode*, and the full Ceph cluster(MON, MGR, and OSD) runs entirely on host servers. In DoCeph, the SmartNIC is switched to *DPU mode*, and the Ceph cluster is instantiated on the DPU. As shown in Figure 3, OSDs execute on the DPU while the host retains only the BlueStore.

The workload is generated with RADOS bench [4] tool using write-only pattern. For each configuration, 16 concurrent clients issue requests with sizes varied across 1 MB, 4 MB, 8 MB, and 16 MB to observe how performance scales with I/O granularity.

Host-side CPU utilization is measured using htop and iostat utilities, sampling every second throughout the benchmark duration. Throughput and latency metrics are captured using RADOS bench's built-in instrumentation. Latency represents the average per-second end-to-end response time for write operations, calculated from all client thread requests during each second of the benchmark. IOPS reflects the average number of write requests completed per second. Each experimental configuration is repeated 5 times, and results are averaged to ensure statistical reliability.

In the following sections, we present experimental results that address these central questions:

- How much host CPU overhead does Ceph's messenger layer impose during normal operation?
- Can DPU offloading significantly reduce host CPU utilization?
- How does DoCeph's performance (latency and throughput) compare to `Baseline` across different I/O request sizes?

5.2 Analysis of Host Resource Overhead

Time breakdown analysis of CPU usage. We analyze the messaging overhead in Ceph while serving I/O requests. For evaluation, we build the Ceph cluster to use two storage nodes and one client, and each storage node runs a single OSD instance. We use RADOS bench as the workload generator, issuing 4MB write operations for 60 seconds. The 1Gbps and 100Gbps configurations used in this section employ different network interfaces. The 1Gbps configuration utilizes a standard Gigabit Ethernet interface, while the 100Gbps configuration uses the BlueField-3 integrated ConnectX-7 network controller. Both configurations run the complete Ceph stack on the host CPU without DPU offloading. Details of the experimental setup are described in Section 5.1.

Figure 5 shows the relative share of total Ceph CPU usage (left axis) attributed to Messenger, ObjectStore, and OSD threads, as well as the corresponding Ceph CPU usage normalized to a single core (right axis) under different network settings. The network environments evaluated are 1Gbps Ethernet and 100Gbps using a DPU in NIC mode. The CPU usage breakdown shown in Figure 5 is obtained using Linux perf profiling tools combined with Ceph's internal thread naming conventions. Messenger threads are identified by the "msgr-worker-" pattern (e.g., msgr-worker-0, msgr-worker-1, msgr-worker-2), ObjectStore operations run in "bstore_" threads, and OSD thread pool operations execute in "tp_osd_tp" threads. These represent mutually exclusive thread categories within Ceph's architecture, eliminating temporal overlap in measurements. Profiling data is collected using perf record during the entire 60-second

Table 1: Testbed specification.

CPU	AMD EPYC 9474F 48-Core Processor
DPU	BlueField-3 integrated ConnectX-7
Memory	DDR5, 256 GB
Storage	Samsung SSD PM893
Network (`Baseline`)	BlueField-3 integrated ConnectX-7 (NIC Mode, Ethernet 100Gbps)
Network (DoCeph)	BlueField-3 integrated ConnectX-7 (DPU Mode, Ethernet 100Gbps)

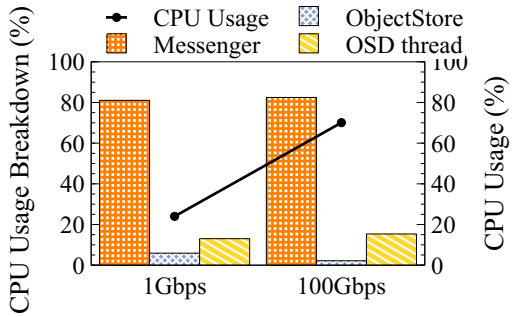

Figure 5: CPU usage breakdown of each component under different network configurations (left axis) and total Ceph CPU usage (right axis). Messenger and ObjectStore represent network processing and storage processing, respectively, while OSD thread represents the operations of threads executed within the OSD.

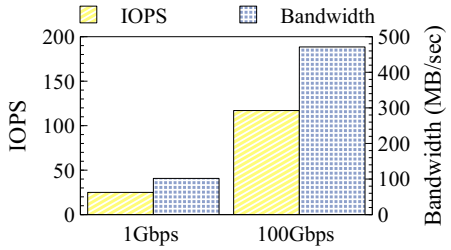

Figure 6: Throughput under 1Gbps and 100Gbps network configurations.

benchmark and analyzed with perf report to extract per-thread CPU usage statistics.

In Figure 5, under the 1Gbps configuration, Messenger, the network layer in Ceph, accounts for approximately 81.05% of total CPU usage, while ObjectStore and OSD threads exhibit significantly lower usage. In Ceph, Messenger handles network transmission and reception through its dedicated worker threads, concentrating CPU-intensive tasks such as TCP send/receive operations in this layer. ObjectStore (e.g., BlueStore) is responsible for actual data persistence, while OSD threads act as an intermediary that coordinates request dispatching and state management.

We then increased the network bandwidth to 100Gbps to verify whether link speed itself was the bottleneck. However, even at 100Gbps, the Messenger component still accounted for 82.48% of total CPU usage, a negligible difference compared to the 1Gbps case. While overall CPU utilization increased from 24% to 70.08% because of higher throughput (Figure 6), Messenger's CPU utilization remains nearly constant at around 80% across both configurations. It indicates that the bottleneck does not lie in the link capacity itself, but in the CPU-bound network processing path, particularly in Messenger's TCP/IP stack handling and packet-processing overhead.

Context Switch Overhead Analysis. These results show that CPU cycles are heavily concentrated in the network layer (Messenger). The frequent context switches caused by TCP/IP stack calls act as one of the main factors exacerbating the bottleneck. To quantify this effect, we measured the average number of context switches in Messenger and ObjectStore under the same workload.

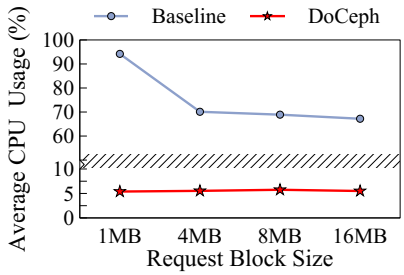

Figure 7: Host CPU usage comparison between `Baseline` and DoCeph across different request sizes (1MB to 16MB).

Table 2: Comparison of the number of context switches between Messenger and ObjectStore components.

	Messenger	ObjectStore
Context Switches	7475	751

Table 2 shows that the number of context switches in Messenger is 9.95x higher than in ObjectStore. This is because Messenger repeatedly invokes TCP/IP stack calls that involve user–kernel mode transitions, kernel buffer copies, and socket event processing, all of which result in far more frequent context switches than ObjectStore. These overheads increase network processing latency and waste CPU resources, further amplifying the bottleneck. In other words, the CPU cycles are disproportionately concentrated in Messenger, whereas BlueStore and OSD threads consume relatively fewer resources. Therefore, it is necessary to decouple and offload Messenger-related execution paths from the host CPU. Such an approach could significantly reduce CPU overhead and allow the host to dedicate more cycles to storage back-end operations, which is critical for improving overall throughput and scalability.

5.3 CPU Utilization under Write Workloads

We first investigated the effectiveness of DoCeph in reducing the CPUs consumed to perform write benchmark. Figure 7 shows the average CPU utilization by write request size. The experimental results reveal that `Baseline` exhibited relatively high CPU utilization of 94.2%, 70.1%, 68.9%, and 67.2% for each request size, while DoCeph recorded significantly lower CPU utilization of 5.5%, 5.75%, 5.53%, and 5.39%, respectively. Consequently, DoCeph achieved host CPU savings of 94.2%, 91.8%, 91.9%, and 92% compared to `Baseline` for each request size. While `Baseline`'s CPU utilization decreases slightly as request size increases, it still maintains a high level, whereas DoCeph consistently maintains a low level of 5-6% across all request sizes.

DoCeph's CPU reduction stems from offloading the most CPU-intensive OSD path in Ceph to the DPU. OSD consumes significant CPU resources by performing tasks such as messenger handling, message decoding and dispatching, and executing worker threads for replication and recovery within the Ceph cluster. However, in DoCeph, only the BlueStore I/O path remains on the host to handle disk operations and metadata processing. It structurally eliminates CPU consumption from OSD thread scheduling and networking.

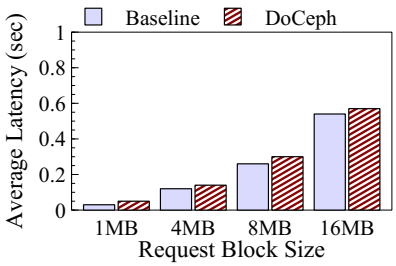

Figure 8: Average latency comparison between `Baseline` and DoCeph across different request sizes (1MB to 16MB).

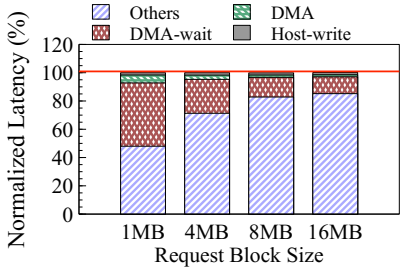

Figure 9: Normalization of average latency time breakdown.

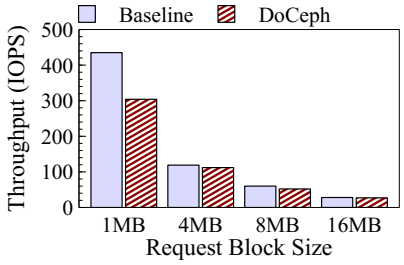

Figure 10: Average throughput comparison between `Baseline` and DoCeph across different request sizes (1MB to 16MB).

Furthermore, by separating communication between the DPU and host into RPC (control plane) and DOCA DMA (data plane), host CPU intervention during data movement is minimized, alleviating kernel network stack and memory copy overheads. Through this approach, DoCeph maintains low and flat CPU utilization, effectively reducing the high CPU consumption observed in `Baseline`.

5.4 Throughput and Latency

To evaluate both the CPU savings and the performance benefits of DoCeph, we measured average latency and throughput under varying write request sizes. Latency is defined as the average time elapsed from the start of a client's single write request until the receipt of a completion response. As shown in Figure 8, DoCeph exhibits higher latency than `Baseline` across all block sizes. For 1MB requests, DoCeph shows a latency of 0.05 seconds compared to `Baseline`'s 0.03 seconds, a difference of approximately 67%. At 16MB, this difference decreases to about 6%, with DoCeph at 0.57 seconds and `Baseline` at 0.54 seconds. To better understand these latency differences, Table 3 provides a detailed time analysis of DoCeph's performance.

Table 3: Average latency time breakdown(sec) of DoCeph.

Request Block Size	1MB	4MB	8MB	16MB
Host write	0.0008	0.0024	0.0046	0.0084
DMA	0.0028	0.0042	0.00523	0.00846
DMA-wait	0.0224	0.0336	0.0418	0.0676
Others	0.024	0.0998	0.24837	0.48554
Total Avg.Latency	0.05	0.14	0.3	0.57

DMA represents the actual data transfer time, Host-write indicates the time taken to write data to BlueStore on the host, DMA-wait shows the waiting time that occurs due to serial DMA transfers. Others encompass various processing times including DPU-resident OSD thread operations, messenger layer activities (message encoding/decoding, connection management, heartbeat processing, etc.), replication coordination, and communication overhead (message serialization, ACK waiting, etc.).

DoCeph uses DOCA DMA for direct memory transfers without CPU intervention, but hardware constraints limit single DMA transfers to 2MB, requiring large requests to be split into multiple segments for serial transmission. To overcome this limitation, DoCeph implements memory region reuse and the pipelining mechanism shown in Figure 4, allowing data preparation and transfer to overlap. Analysis shows that the main latency differences between `Baseline` and DoCeph stem from DMA-wait time and performance differences between the DPU and host CPU. Figure 9 shows the normalized time breakdown for Table 3.

For small block sizes like 1MB, DMA-wait accounts for approximately 44.8% of the total latency, representing significant overhead. However, as block size increases, the relative proportion of DMA-wait decreases to about 11.9% at 16MB. Notably, as block size increases, the pipelining effect is maximized, reducing the proportion of DMA-wait. While absolute DMA-wait time increases from 0.0224 to 0.0676 seconds as request size grows 16-fold, its relative contribution decreases dramatically from 44.8% to 11.9% of total latency. Larger requests are divided into multiple segments, and the transmission of the next segment can begin before the processing of the previous segment is complete, enabling efficient overlap. Due to this pipelining effect, the performance gap between DoCeph and `Baseline` narrows significantly as block size increases, with latency overhead reducing from 67% at 1MB to just 6% at 16MB.

Figure 10 shows the average throughput (IOPS) for each write request size. For 1MB writes, DoCeph achieves 304 IOPS while `Baseline` reaches 435 IOPS, indicating that DoCeph's throughput is approximately 30% lower. However, as block size increases, the throughput gap between the two systems narrows significantly. At 4MB, the IOPS values for DoCeph and `Baseline` are 112 and 119 respectively, with DoCeph performing about 6% lower. For 8MB requests, DoCeph delivers 52 IOPS compared to `Baseline`'s 60 IOPS, demonstrating approximately 13% lower throughput. At 16MB, DoCeph's 27 IOPS is slightly lower than `Baseline`'s 28 IOPS by about 4%.

Although the 2MB size limitation of DMA transfers requires large blocks to be divided into multiple segments for serial transmission, DoCeph largely overcomes these constraints through optimizations

including memory region reuse, segment pipelining, and streaming write mechanisms. For small objects like 1MB, performance is degraded due to significant DMA-wait time overhead and limited pipelining effects. In contrast, for objects larger than 1MB, pipelining reduces DMA-wait overhead, and CPU-intensive messenger operations are offloaded to the DPU, allowing the host CPU to focus exclusively on BlueStore operations. Consequently, DoCeph provides throughput comparable to `Baseline` for large objects while significantly reducing host CPU utilization.

In summary, DoCeph leverages the DPU's ARM cores to offload Ceph OSD tasks and significantly reduce host CPU usage. Although latency is higher due to offload-induced coordination, DoCeph delivers substantial throughput gains for large-block workloads and demonstrates decreasing latency as block size increases.

5.5 Read Path Performance Discussion

While this paper focuses on write operations, the read path follows a symmetric design where client read requests are processed through the same proxy architecture. When clients issue read requests, the DPU uses ProxyObjectStore to send the necessary request metadata to the host via DMA, and after the host-side BlueStore performs the read operation, the requested data is transferred back to the DPU via DMA before being transmitted to clients.

Based on write performance patterns, DoCeph is expected to show similar convergence behavior at large block sizes for read operations, potentially with even better relative performance since reads avoid replication coordination overhead. The bidirectional DMA architecture should provide consistent throughput benefits while maintaining the CPU offloading advantages demonstrated in write operations.

6 Related Works

Recent research has explored the use of Data Processing Units (DPUs) to offload storage tasks from host CPUs and improve data center efficiency. In disaggregated architectures, DDS [25] utilizes DPUs to process read operations directly along the data path, reducing latency and host involvement. HiDPU [26] proposes a hybrid indexing structure tailored for DPUs to minimize communication overhead and manage memory constraints by segmenting index metadata across the host and the DPU. Other systems explore DPU-based key-value stores. LEED [7] addresses resource imbalance in SmartNIC-based JBOF arrays by re-architecting the I/O and memory interface to optimize KV operations. OS2G [9] offloads the object storage client onto the DPU and enables direct data transfer to GPUs using GPUDirect, reducing data movement overhead in deep learning workloads.

Hardware acceleration within DPUs has also been studied. PEDAL [12] leverages compression/decompression engines in BlueField DPUs for real-time HPC data streaming, while INEC [19] integrates erasure coding with RDMA operations for efficient NIC-level execution. Fuyao[13] decouples control and data flows in serverless computing and leverages DPUs to enable sub-millisecond direct data transfer between functions, significantly reducing latency and CPU usage. DFlush [6] focuses on offloading flush operations from LSM-based key-value stores to DPUs, proposing a 3D-parallel data plane and adaptive control plane to manage pipeline and resource contention efficiently. In distributed training workloads, FSDP acceleration has been achieved by offloading collective communication logic to SmartNICs [11], utilizing hardware multicast and datapath accelerators to reduce CPU load and enable scalable bandwidth-aware operations.

While prior work has largely focused on offloading specific storage functions or client-side components, our work takes a step further by offloading the entire storage server logic (e.g., Ceph OSD) to the DPU. By doing so, we aim to minimize host CPU involvement across the full data path, enabling true in-path storage offloading with centralized data placement logic retained on the host.

7 Conclusion

In this work, we presented DoCeph, a novel DPU-offloaded Ceph architecture that decouples the OSD from the host's object store and transparently mediates all backend interactions between DPU and host. DoCeph executes all core OSD logic—including request handling, replication, and recovery—on the DPU, and retains only the BlueStore backend on the host. As a result, DoCeph significantly reduces host CPU utilization and eliminates the dominant bottleneck caused by Ceph's messenger layer. Our evaluation demonstrates that DoCeph reduces host CPU usage by up to 92% compared to baseline Ceph, while maintaining stable throughput and latency, especially for large-block write workloads.

Acknowledgments

This research was partially supported by the Korea Institute of Science and Technology Information (KISTI) (No. K25L1M2C2-01, NTIS No. 2710087345) and by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No. RS-2024-00453929). This research also used resources of the Oak Ridge Leadership Computing Facility, located at the National Center for Computational Sciences at the Oak Ridge National Laboratory, which is supported by the Office of Science of the DOE under Contract DE-AC05-00OR22725.

References

[1] AMD. 2022. AMD Pensando DPU Technology: Front-end networking powering modern data centers. https://www.amd.com/en/products/data-processing-units/pensando.html.
[2] Broadcom. 2024. Broadcom Stingray SmartNIC Accelerates Baidu Cloud Services. https://www.broadcom.com/company/news/product-releases/53106.
[3] Ceph. 2016. FileStore Config Reference. https://docs.ceph.com/en/reef/rados/configuration/filestore-config-ref/.
[4] Ceph. 2016. RADOS Object Storage Utility. https://docs.ceph.com/en/latest/man/8/rados/.
[5] Alibaba Cloud. 2022. A Detail Explanation about Alibaba Cloud CIPU. https://www.alibabacloud.com/blog/a-detailed-explanation-about-alibaba-cloud-cipu_599183.
[6] Chen Ding, Kai Lu, Quanyi Zhang, Zekun Ye, Ting Yao, Daohui Wang, Huatao Wu, and Jiguang Wan. 2025. DFlush: DPU-Offloaded Flush for Disaggregated LSM-based Key-Value Stores. *Proc. ACM Manag. Data* 3, 3, Article 147 (June 2025), 28 pages. https://doi.org/10.1145/3725284
[7] Zerui Guo, Hua Zhang, Chenxingyu Zhao, Yuebin Bai, Michael Swift, and Ming Liu. 2023. LEED: A Low-Power, Fast Persistent Key-Value Store on SmartNIC JBOFs. In *Proceedings of the ACM SIGCOMM 2023 Conference* (New York, NY, USA) *(ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 1012–1027. https://doi.org/10.1145/3603269.3604880
[8] Intel. 2022. Intel Infrastructure Processing Unit(Intel IPU). https://www.intel.com/content/www/us/en/products/details/network-io/ipu.html.
[9] Zhen Jin, Yiquan Chen, Mingxu Liang, Yijing Wang, Guoju Fang, Ao Zhou, Keyao Zhang, Jiexiong Xu, Wenhai Lin, Yiquan Lin, Shushu Zhao, Wenkai Shi, Zhenhua

He, Shishun Cai, and Wenzhi Chen. 2025. OS2G: A High-Performance DPU Offloading Architecture for GPU-based Deep Learning with Object Storage. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 750–765. https://doi.org/10.1145/3676641.3716265

[10] Arjun Kashyap, Yuke Li, Darren Ng, and Xiaoyi Lu. 2025. Understanding the Idiosyncrasies of Emerging BlueField DPUs. (2025).

[11] Mikhail Khalilov, Salvatore Di Girolamo, Marcin Chrapek, Rami Nudelman, Gil Bloch, and Torsten Hoefler. 2024. Network-Offloaded Bandwidth-Optimal Broadcast and Allgather for Distributed AI *(SC '24)*. IEEE Press, Article 103, 17 pages. https://doi.org/10.1109/SC41406.2024.00109

[12] Yuke Li, Arjun Kashyap, Weicong Chen, Yanfei Guo, and Xiaoyi Lu. 2024. Accelerating Lossy and Lossless Compression on Emerging BlueField DPU Architectures. In *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 373–385. https://doi.org/10.1109/IPDPS57955.2024.00040

[13] Guowei Liu, Laiping Zhao, Yiming Li, Zhaolin Duan, Sheng Chen, Yitao Hu, Zhiyuan Su, and Wenyu Qu. 2024. FUYAO: DPU-enabled Direct Data Transfer for Serverless Computing. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 431–447. https://doi.org/10.1145/3620666.3651327

[14] Marvell. 2021. Marvell Octeon Data Processing Units. https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-octeon-10-dpu-platform-white-paper.pdf.

[15] Mircrosoft. 2024. Enhancing infrastructure efficiency with Azure Boost DPU. https://techcommunity.microsoft.com/blog/azureinfrastructureblog/enhancing-infrastructure-efficiency-with-azure-boost-dpu/4298901.

[16] Nvidia. 2021. Nvidia Bluefield-3 DPU: Programmable data center infrastructure on-a-chip. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf.

[17] NVIDIA. 2024. DOCA Communication Channel. https://docs.nvidia.com/doca/sdk/doca+comch/index.html.

[18] Amazon Web Service. 2024. AWS Nitro System. https://aws.amazon.com/ec2/nitro/.

[19] Haiyang Shi and Xiaoyi Lu. 2020. INEC: Fast and Coherent In-Network Erasure Coding. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–17. https://doi.org/10.1109/SC41405.2020.00070

[20] Haodong Tang, Jian Zhang, and Fred Zhang. 2018. Accelerating Ceph with RDMA and NVMe-oF. In *Proc. 14th Annu. Workshop OpenFabrics Alliance*. 1–27.

[21] SAGE Weil. 2017. Bluestore: A new storage backend for ceph one year in. *March2017. Retrieved August* 19 (2017), 2018.

[22] Sage Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. 2006. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI'06)*. 307–320.

[23] Sage A Weil, Scott A Brandt, Ethan L Miller, and Carlos Maltzahn. 2006. CRUSH: Controlled, scalable, decentralized placement of replicated data. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. 122–es.

[24] Sage A Weil, Andrew W Leung, Scott A Brandt, and Carlos Maltzahn. 2007. Rados: a scalable, reliable storage service for petabyte-scale storage clusters. In *Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing'07*. 35–44.

[25] Qizhen Zhang, Philip A. Bernstein, Badrish Chandramouli, Jiasheng Hu, and Yiming Zheng. 2024. DDS: DPU-Optimized Disaggregated Storage. 17, 11 (July 2024), 3304–3317. https://doi.org/10.14778/3681954.3682002

[26] Wenbin Zhu, Zhaoyan Shen, Qian Wei, Renhai Chen, Xin Yao, Dongxiao Yu, and Zili Shao. 2025. HiDPU: A DPU-Oriented Hybrid Indexing Scheme for Disaggregated Storage Systems. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*. USENIX Association, Santa Clara, CA, 271–285. https://www.usenix.org/conference/fast25/presentation/zhu